Università di Padova Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica
Tesi di Laurea

Piattaforma di supporto Open Source per Android su Architettura ARM11

Relatore: **Chiar.mo Prof. Lorenzo Vangelista**

Laureando: **Alberto Panizzo**

Anno Accademico 2008-2009

A Elena, Giancarlo, Marco
e tutti, ma proprio Tutti
che mi sono accanto.

Indice

Indice **1**

Introduzione **3**

1 Sistemi Embedded e Mobile Devices **5**

1.1 Mobile Devices 6

1.2 Sistemi Operativi per Dispositivi Mobili 7

1.2.1 Architettura di sistema per i moderni Mobile OS . . . 8

1.3 Il mercato dei Sistemi Operativi per dispositivi mobili 9

1.4 Ruolo dell'Open source 12

2 Android **15**

2.1 Licenza della piattaforma 15

2.2 Architettura 16

2.2.1 Kernel Level 16

2.2.2 Framework Libraries 21

2.2.3 Application Framework 25

2.2.4 Il file Manifest 27

2.2.5 Attività e componenti affini 28

2.2.6 Ciclo di vita dei componenti 28

2.2.7 Processi e ciclo di vita dei componenti 33

2.3 Android SDK 34

2.3.1 Personalizzare le versione di Android eseguita dall'emulatore 35

2.4 Android e il Business 36

3 Microarchitettura ARM11 **39**

3.1 Architettura ARMv6 40
3.2 Innovazioni nella microarchitettura ARM11 43
3.3 Il Core ARM1136JF-S 45

4 L'Application Processor Freescale i.MX31L **47**

4.1 L'Application Processor 47
4.2 Il Core di Elaborazione 49
4.3 La gestione degli interrupt 49
4.4 Interfaccia verso la memoria esterna 50
4.5 Clock e Power Management 50
4.6 Multiplexing, GPIO, e Pad Control 51
4.7 Interfaccia AIPS verso il bus di espansione 51
4.8 Shared Peripheral Bus Arbiter (SPBA) 51
4.9 Smart Direct Memory Access Controller 52
4.10 Timers GPT ed EPIT 53
4.11 Interfacce seriali UART 55
4.12 Interfaccia I^2C 55
4.13 Interfaccia USB 55
4.14 Secured Digital Host Controller (SDHC) 57
4.15 Interfaccia PCMCIA 58
4.16 Il modulo Watchdog 58
4.17 Real Time Clock 58
4.18 Immagini, Video e Grafica 59
4.19 Interfaccia Audio 61

5 Atmark Armadillo 500 **63**

5.1 L'hardware della board nel dettaglio 64
5.1.1 Armadillo 500 SoM 64
5.1.2 Porte seriali RS232 64

5.1.3 Rete Ethernet 64
5.1.4 Slot SD/MMC 65
5.1.5 Output Video 65
5.1.6 NAND Flash 66
5.1.7 Real Time Clock (RTC) 66
5.1.8 Tasti e LED 66
5.1.9 USB 66
5.1.10 Audio 66
5.2 Il boot loader 67
5.2.1 Canale di comunicazione e controllo 67
5.2.2 Contatti di configurazione 67
5.2.3 Comandi fondamentali 68
5.3 Compatibilità tra revisioni successive del SoM Armadillo 500 70

6 Linux su piattaforma Freescale i.MX31L 71
6.1 Il progetto ARM-Linux 73
6.1.1 Mailing list e gestione delle Patch 73
6.1.2 Organizzazione del codice di supporto per l'architettura ARM 76
6.1.3 Fasi di boot di Linux su architettura ARM 77
6.1.4 Metodi di debug 85
6.1.4.1 Early debug 86
6.1.4.2 Normal Debug 87
6.1.4.3 Driver Debug 88
6.2 Ambiente di sviluppo, la macchina host 89
6.2.1 Posizione del progetto 90
6.2.2 Il kernel 91
6.2.3 Rudimenti di Git per la creazione di patch 92
6.2.4 Cross-Platform Toolchain 94
6.2.5 Configurare e compilare il kernel 98
6.2.6 Creare un root filesystem valido 100

6.2.7 La connessione con la macchina target 103
6.2.8 Scaricare il kernel nella board Armadillo 500 105
6.3 Fasi del porting 106
6.3.1 Early support 106
6.3.2 Console attraverso la porta seriale 112
6.3.3 Rete Ethernet 115
6.3.4 SDHC MMC 120
6.3.5 Output Video 125
6.3.6 Modulo flash NOR 135
6.3.7 Modulo flash NAND 138
6.3.8 GPIO Keyboard 141
6.3.9 RTC 143
6.3.10 USB 146

7 Android su piattaforma Freescale i.MX31L 155
7.1 Il codice sorgente di Android 155
7.1.1 Installare repo 155
7.1.2 Download dei sorgenti 156
7.1.3 Prima build di Android 156
7.2 Ottenere un kernel valido 157
7.2.1 Le patch Android 158
7.2.2 Avanzare la piattaforma i.MX 162
7.2.3 Configurare il kernel Android ottenuto. 164
7.3 Personalizzare il processo di build per la board Armadillo 500 165
7.3.1 Definire un prodotto. 166
7.3.2 Impostzioni board-specific di generazione dello stack. 167
7.3.3 Modificatori di prodotto 170
7.3.4 Generare il root-filesystem definito dal prodotto armadillo500 171
7.4 Problemi e Soluzioni 172
7.4.1 Framebuffer 172

7.4.2 Battery 175
7.4.3 Mouse USB come sistema di tracking 176

8 Risultati **179**

8.1 Piattaforma di supporto per la board Atmark Armadillo 500 nel kernel Linux 179
8.2 Android nella board Atmark Armadillo 500 181

9 Conclusioni **185**

A Memory Map **187**

Elenco degli acronimi **189**

Bibliografia **198**

Introduzione

In un mercato importante come quello dei dispositivi portatili è notevole la crescita dell'offerta Open Source: piattaforme Linux come Maemo, Access, QT extended (ex Qtopia) hanno raggiunto un elevato livello di maturità potendo così competere con le piattaforme proprietarie esistenti. Ciò che frena la diffusione delle piattaforme Open Source è la mancanza di una spinta concreta da parte dei produttori di dispositivi i quali hanno privilegiato finora le più note soluzioni proprietarie.

Android si inserisce in questo contesto con una marcia in più: nasce in seno alla Open Handset Alliance alleanza tra operatori mobili, produttori hardware e compagnie software tra le maggiori al mondo capaci di fornire quella rete di sinergie necessaria per una rapida diffusione globale. Android è uno stack software Open Source completo per la gestione dei dispositivi portatili: basato sul kernel Linux (arricchito con nuove funzionalità IPC e memory sharing), fornisce le librerie necessarie per una user experience evoluta ed applicazioni chiave nell'ambito mobile.

La struttura di questo lavoro di tesi prevede nel Capitolo 1 la contestualizzazione del problema posto nell'ambito dei Sistemi Embedded e più specificatamente in quello dei Mobile Devices. Nel Capitolo 2 vi è una presentazione approfondita del Sistema Operativo Android: del lato tecnico (Architettura, Modello delle applicazioni etc.) e della della filosofia che Android porta con se nella licenza di distribuzione e nel modello di businness applicato. I Capitoli 3 e 4 descrivono del dettaglio l'architettura dell'Application Processor montato nella board Atmark Armadillo 500 presentata a sua volta nel Capitolo 5.

Successivamente si entra nella fase operativa del progetto svolto: nel Capitolo 6 viene trattato lo sviluppo del livello di astrazione software per il kernel Linux a supporto della Board Atmark Armadillo 500. Il codice prodotto è stato sottoposto a review da parte della comunità open source e ora è presente nei sorgenti pubblici del main line kernel. Il Capitolo

7 tratta infine della fusione dello strato software creato con il branch Android del kernel Linux e la generazione di una versione personalizzata della piattaforma Android per la board di sviluppo Armadillo 500.

I Capitoli 8 e 9 terminano la trattazione del presente lavoro di Tesi presentando e commentando i risultati ottenuti.

Capitolo 1

Sistemi Embedded e Mobile Devices

I sistemi embedded hanno ormai consolidato un ruolo importante nella nostra vita quotidiana; rappresentano la classe dei computer progettati per uno scopo specifico, al contrario dei comuni PC progettati per l'esecuzione delle più svariate applicazioni.

Esistono diversi tipi di sistemi embedded caratterizzati da diversi gradi di specificità delle funzionalità offerte in relazione agli obbiettivi progettuali, di affidabilità e prevedibilità richieste: dai sistemi di controllo per macchine industriali ai dispositivi di sicurezza delle automobili (ABS EBD etc.), dai dispositivi per il networking (switch, access point) ai dispositivi mobili di ultima generazione.

Tutti questi sistemi sono sviluppati con lo scopo di migliorare la qualità della vita, da un lato automatizzando in modo intelligente ed affidabile compiti critici e dall'altro fornendo funzionalità sempre più ricche per l'utente.

Una definizione generale è:

> I *sistemi embedded* sono sistemi di calcolo con una forte integrazione tra hardware e software e progettati per garantire ottime prestazioni nello svolgimento di un compito specifico. La parola inglese "embedded" significa incluso e riflette il fatto che questi sistemi sono spesso una parte integrante di un sistema molto più grande, che se è incorporato in qualche dispositivo viene chiamato, anch'esso, sistema embedded; un sistema embedded può essere composto da più sistemi embedded

interagenti tra loro.

Particolare attenzione verrà posta in questo elaborato ai Dispositivi Mobili (Mobile Devices); si tratta di una categoria dei sistemi embedded che ha visto un notevole sviluppo in questi ultimi anni grazie alla spinta tecnologica, portando all'aumento delle funzionalità offerte ed alla nascita di prodotti "ibridi" per una risposta migliore alle esigenze dell'utente.

1.1 Mobile Devices

> Vengono classificati *Mobile Devices* tutti quei dispositivi elettronici a microprocessore di dimensioni ridotte, dotati di una forma di alimentazione integrata che ne permetta il funzionamento slegato dalla rete elettrica. Solitamente possiedono uno schermo ed una o più modalità di input (tastiera, touch).

Di questa categoria fanno parte dispositivi di dimensioni diverse, seppur portabili, che vanno dai PC-Notebook ai telefoni portatili (detti *handheld*).

In relazione alle dimensioni stanno le funzionalità offerte in misura dipendente dalla tecnologia interna del dispositivo: la ricchezza di funzionalità nel tempo è andata aumentando grazie all'evoluzione tecnologica nel campo dell'elettronica che ha permesso di passare in una decina d'anni, per il form factor hand-held, da prodotti mono-funzione (telefono, fotocamera, gaming-console, media player) a smartphone all-in-one capaci di soddisfare le esigenze di un utente medio con un unico prodotto.

L'evoluzione dei dispositivi mobili però non ha decretato "ingombranti" prodotti quali notebook o "inutili" i dispositivi specifici come le fotocamere digitali; è stato portato avanti invece un processo di affinamento delle qualità specifiche del singolo prodotto in relazione alle seguenti misure:

Portabilità - Intesa come grado di facilità di trasporto in relazione alle dimensioni: più è piccolo il prodotto meglio può essere trasportato e quindi più frequentemente può essere utilizzato.

Produttività - Intesa come grado di complessità del lavoro eseguibile dall'utente in relazione ai metodi di input ed alla qualità del workspace: più è piccolo o difficilmente manovrabile il prodotto minore sarà la produttività dell'utente.

Connettività - Indica la capacità del prodotto di connettersi alle reti di comunicazione: maggiori sono le tecnologie di rete supportate e la loro qualità, maggiore sarà la possibilità di connessione con la rete globale.

Multimedialità e divertimento - Indica la capacità del prodotto di riprodurre e/o registrare e/o elaborare contenuti multimediali e di intrattenere l'utente attraverso giochi o applicazioni per il tempo libero.

Costo - maggiori saranno le precedenti misure maggiore sarà il costo del dispositivo.

Allora le dimensioni di un notebook, pur sacrificando la portabilità, permettono alte Produttività, Connettività e Multimedialità mentre gli smartphone di nuova generazione guadagnano notevolmente in Portabilità ma perdono necessariamente in Produttività date le interfacce di input e output meno agevoli.

E' per ricercare un rapporto migliore tra queste misure che sono nate le categorie *Netbook* e *Smartbook*, il primo un notebook più portabile di notevole successo, mentre il secondo uno smartphone più produttivo.

Una costante dell'evoluzione dei dispositivi mobili è la spinta verso lo status *"always connected"* per il quale il dispositivo è sempre connesso (o ha sempre la possibilità di connettersi) alla rete globale attraverso tecnologie di rete di qualità sempre maggiore. Questa capacità rende i dispositivi mobili fonti di informazioni preziose per l'utente, rende pervasivo il concetto di presenza, permette forme evolute di social networking e connettività ma permette inoltre, ad aziende che operano nella rete globale, di aumentare il bacino d'utenza per i propri servizi.

1.2 Sistemi Operativi per Dispositivi Mobili

Alcuni fattori che caratterizzano i dispositivi mobili hanno costituito (e costituiscono ancora) vincoli importanti nello sviluppo dell'intero stack software che li anima: le dimensioni ridotte ed il ridotto budget energetico dei dispositivi *hand-held* limitano le risorse computazionali a disposizione (capacità di calcolo, dimensione della memoria) e ridefiniscono le modalità di I/O.

Per le prime generazioni di device hand-held questi vincoli erano così stringenti che il software di gestione del dispositivo doveva essere svi-

luppato ad-hoc dalle aziende produttrici senza la possibilità di realizzare i più semplici paradigmi che caratterizzano i sistemi operativi odierni (multitasking, gestione condivisa delle risorse, astrazione).

L'evoluzione tecnologica ha prodotto un rilassamento dei vincoli sopra esposti permettendo così la realizzazione di stack software sempre più completi: modulari, capaci di supportare gli standard attuali di comunicazione e interoperabilità, forniti di framework per lo sviluppo di applicazioni verso una maggiore ricchezza di funzionalità offerte, orientati ai contenuti ed al valore aggiunto che questo possa dare.

1.2.1 Architettura di sistema per i moderni Mobile OS

L'architettura dei moderni sistemi operativi per dispositivi mobili (MobileOS) si presenta come in figura 1.1.

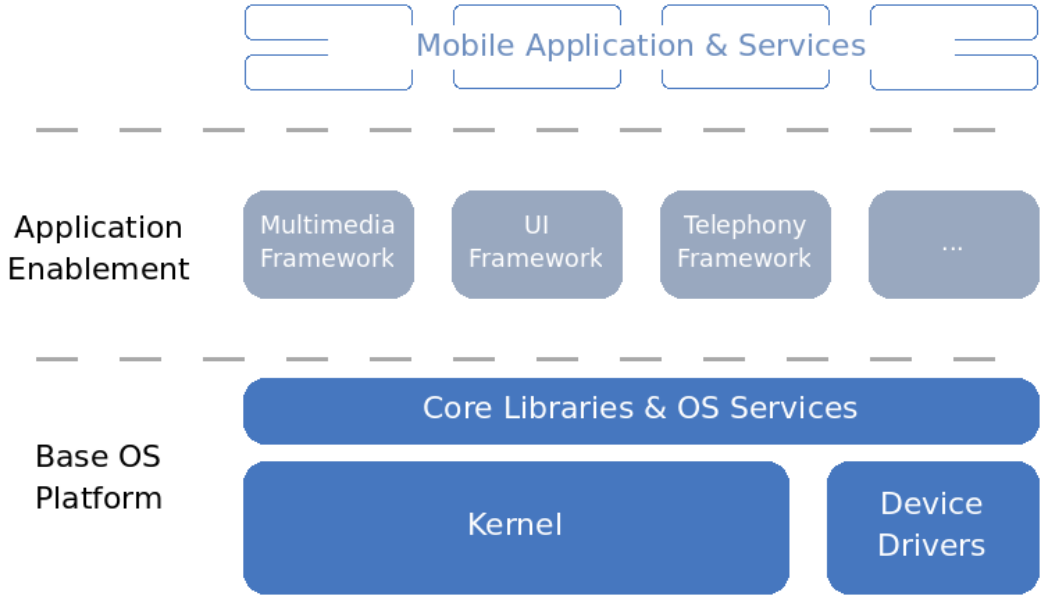


Figura 1.1: Stack software completo per dispositivi mobili.

Base OS Platform

La Base OS Platform è simile all'architettura dei vicini Sistemi Operativi General Purpose (GPOS) ma ottimizzata per i vincoli presenti nei dispositivi hand-held:

Kernel: deve gestire in modo efficiente le limitate risorse computazionali, di memoria ed energia disponibile. Lo scheduler implementa funzionalità real-time, vengono implementati file system studiati appositamente per gestire i dispositivi di storage disponibili ed il power management deve essere pervasivo ed efficace.

Device Drivers: data la continua evoluzione tecnologica, la quantità di nuovi dispositivi da gestire attraverso driver è notevole. Sforzi si stanno facendo per standardizzare interfacce verso moduli hardware omogenei.

Librerie di Sistema: ottimizzate nelle dimensioni, se sviluppate in modo da implementare standard aperti (POSIX) danno l'opportunità agli strati superiori di riutilizzare moduli software già esistenti e maturi.

Application Enablement

A questo livello di astrazione vengono implementati i moduli software che caratterizzano il Mobile OS : dalla user interface alla gestione della telefonia, dal framework multimediale (audio, video, 3D) alla gestione delle tecnologie di comunicazione (Bluetooth, WiFi, IrDA ..) passando per il framework Web, sono le librerie che definiscono le capacità funzionali della piattaforma, le potenzialità.

Mobile Application & Services

Sono le applicazioni vere e proprie utilizzabili dall'utente. Solitamente un Mobile OS è fornito con un set di applicazioni di base come Telephone Call, Organizer, Calendar, Media Player etc.. integrate successivamente anche da sviluppatori di terze parti per mezzo di Software Development Kit oppure attraverso framework appositi come Java VM.

1.3 Il mercato dei Sistemi Operativi per dispositivi mobili

Da una ricerca Gartner pubblicata ad Agosto 2009[34] (riassunta in figura 1.2) è possibile notare come nel secondo quarto dello stesso anno si sono affermati nel mercato i più diffusi Mobile OS.

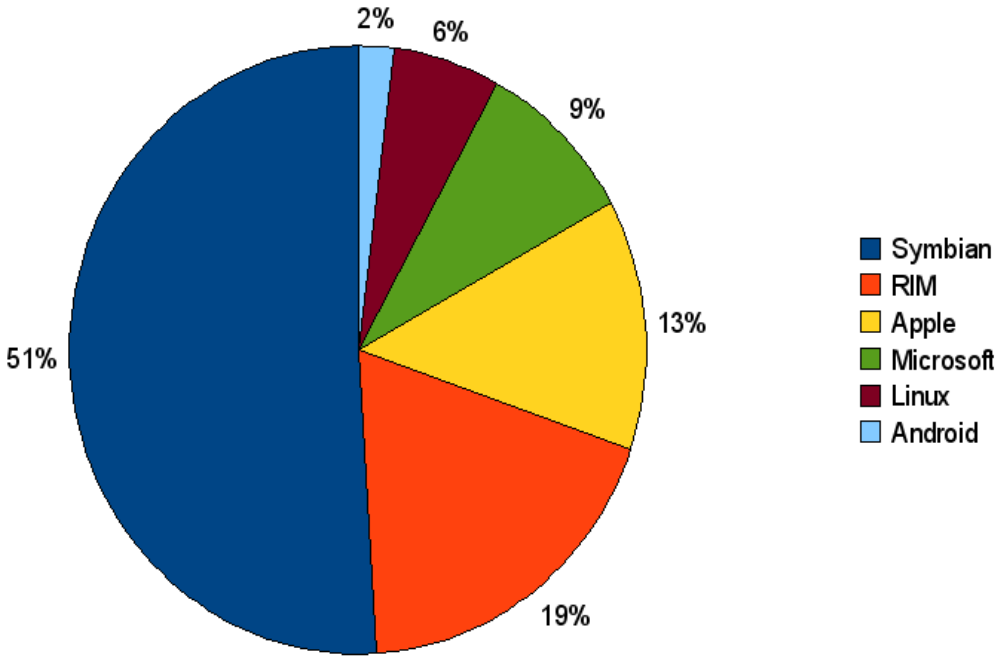


Figura 1.2: Percentuale di affermazione nel Mercato dei maggiori Mobile OS.

Symbian OS di Symbian Ltd - 51%

Nato da una collaborazione tra *Ericsson*, *Nokia*, *Motorola*, e *Psion* concretizzata nella società *Symbian Ltd* nel 1998. Discende dal Sistema operativo EPOC sviluppato dalla sola *Psion*.

Nel 2008 Nokia ha deciso per l'acquisizione dell'intera Symbian Ltd per la costituzione della fondazione non-profit *Symbian Foundation* alla quale affidare l'apertura del codice dell'intera piattaforma con licenza open-source Eclipse Public License (EPL) da completare entro il 2010.

Il sistema operativo Symbian non include lo strato *Application Enablement* così Nokia, Ericsson, ed altre hanno sviluppato al di sopra di Symbian OS, user interface e framework di sviluppo software personali (Nokia *S60 S80 S90*, Ericsson *UIQ*, NTT DoCoMo *MOAP*) non compatibili tra di loro.

BlackBerry OS di RIM - 18,7%

Sistema Operativo studiato appositamente per il mercato business. Il punto di forza è proprio la quantità di applicazioni pensate per i manager supportate da un'infrastruttura backend completa ed interoperabilità con i più comuni software enterprise level.

Nel dicembre 2007 è stato istituito il BlackBerry App World Store dove si possono trovare applicazioni anche di terze parti.

iPhone OS di Apple Inc. - 13,3%

Mobile OS completo, derivato dal sistema operativo *Mac OS X*, che anima l'iPhone e iPod Touch. Pensato innanzitutto per il mercato user presenta una user interface studiata per colpire l'utente e per essere facile da utilizzare.

La diffusione non è stata importante fino al rilascio della versione 2.0 in luglio 2008 con il conseguente rilascio dell'SDK per lo sviluppo di applicazioni da parte di terze parti: sono state implementate applicazioni business level ed è stato aperto L'App Store, un marketplace dove ad ora è possibile trovare ogni genere di applicazione immaginabile.

Windows Mobile di Microsoft - 9%

Basato sul Sistema Operativo *Windows CE*, la piattaforma Windows Mobile è il prodotto di Microsoft per questo settore. Simile al BlackBerry OS come modello di business permette una naturale interoperabilità con i software enterprise level Microsoft (Exchange etc).

Piattaforme Linux - 6%

Il sistema operativo Linux viene utilizzato da diversi vendors come base per lo sviluppo delle proprie Mobile platforms pressoché incompatibili tra di loro. Alcuni protagonisti in questo mercato sono:

Nokia: Ha dato spunto e mantiene diversi progetti open-source, la piattaforma principale è *Maemo* già matura e montata su diversi device presenti nel mercato. Con l'acquisizione di *Trolltech* nel 2008 Nokia ha preso le redini della piattaforma *Qtopia* (ora *Qt Extended*) e più

importante dell'intero sviluppo delle librerie grafiche *Qt* sulle quali è prevista la migrazione della piattaforma Maemo.

Motorola: Prima azienda ad aver portato Linux nei mobile devices specie nel mercato asiatico con la piattaforma *MOTOMAGX* basata su *MontaVista Linux*. Successivamente nel 2007 assieme a *NEC*, *NTT DoCoMo*, *Panasonic Mobile Communications*, *Samsung Electronics*, e *Vodafone* ha fondato la *LiMo Foundation* con l'obbiettivo di sviluppare una piattaforma Linux-based globalmente competitiva e sviluppare standard di interoperabilità tra le diverse piattaforme Linux.

Access: Azienda giapponese che nel 2005 ha acquisito con *PalmSource* la piattaforma *PalmOS* attraverso la quale ha sviluppato il proprio Mobile OS *ACCESS Linux Platform*. Access fa parte della LiMo Foundation e la sua piattaforma ne è conforme alle specifiche di interoperabilità.

Android - 2%

Android è la piattaforma open-source Linux-derived studiata in questo lavoro di tesi. Supportata da *Google* assieme a diverse aziende importanti di sviluppo hardware e software ed operatori mobili riuniti nella *Open Handset Alliance*, grazie alla spinta di questo ecosistema, al modello di sviluppo adottato ed alla libertà di intervento promessa alla comunità, molti operatori del settore pensano ad Android come una piattaforma con grosse potenzialità di crescita.

1.4 Ruolo dell'Open source

Ciò che rende interessante il paradigma open-source applicato ai Mobile OS è la possibilità di creare un luogo di incontro tra un ecosistema aziendale importante e comunità di liberi sviluppatori capace, attraverso il resource sharing, di velocizzare lo sviluppo ed il deployment della piattaforma.

Le velocità del mercato con le quali le aziende Independent Software Vendors (ISV) devono confrontarsi, la quantità di nicchie locali del mercato globale da soddisfare e la notevole mole di lavoro necessaria per mantenere ed evolvere continuamente un intero stack software per mobile devices costituiscono fattori killer per piattaforme proprietarie.

Solamente leader mondiali del settore hanno le capacità economiche ed eredità software tali da reggere la sfida (anche se con qualche acciacco vedi le difficili interfacce Microsoft ed il lento sviluppo dell'iPhoneOS) mentre nuove società o alleanze anche tra i più grossi produttori di dispositivi non sarebbero in grado di mantenere il ritmo di sviluppo (fra tutti Symbian OS, nato dall'alleanza tra i più grossi vendors, ora rilasciato con licenza open-source).

Il software e più ampiamente il paradigma open-source permettono alle nuove piattaforme di competere nelle sfide che il mercato offre: utilizzare il sistema operativo Linux come Base System OS significa avere alte prestazioni (in termini di tempi di risposta e throughput), supporto per una grande varietà di hardware (CPU, periferiche, board specifiche), tool ed utility libere per lo sviluppo; un sistema altamente personalizzabile e ben documentato con una quantità di librerie software già mature non indifferente.

Il software open-source esistente però non copre interamente i bisogni dell'intero stack Mobile OS e la tendenza della comunità open a sviluppare maggiormente funzionalità o supporti che interessano un numero rilevante di componenti della stessa comunità, non aiuta a colmare i gap esistenti.

E' in questa attività che lo sviluppo aziendale può dare quel valore aggiunto che caratterizzerà la propria piattaforma. E' generalmente accettata la barriera detta *value line* (in figura 1.3): una linea di demarcazione tra le componenti software più adatte ad uno sviluppo di tipo open-source e moduli che costituiscono valore aggiunto portato dall'azienda sviluppatrice.

Nell'ultimo periodo sono stati formati consorzi industriali come *OSDL*, *Mobile Linux Initiative (MLI)*, *Consumer Electronics Linux Forum (CELF)*, *Linux Phone Standards Forum (LiPS)* e *Linux Mobile Foundation (LiMo)* per identificare e colmare le lacune di Linux ed ecosistema *FOSS (Free and Open Source Software)* verso una piattaforma mobile comune e standard di interoperabilità con quelle esistenti.

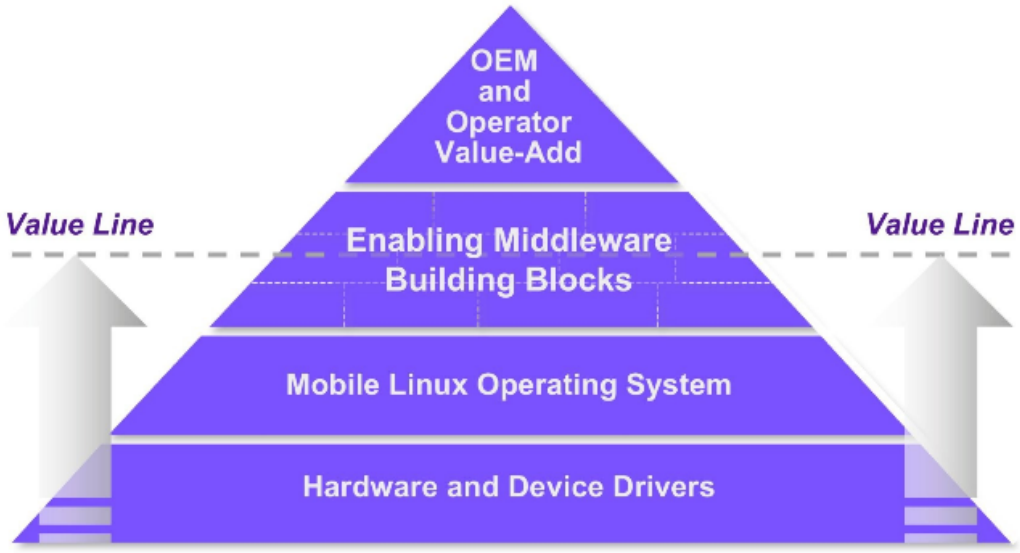


Figura 1.3: Posizione della value line all'interno dello stack software di un Mobile OS. La rappresentazione a piramide da un'indicazione della quantità di ore uomo richieste nello sviluppo dei vari layer nello stack.

Capitolo 2

Android

Sviluppato inizialmente all'interno di Google e promosso da una delle maggiori alleanze del settore, la Open Handset Alliance[1], Android è una piattaforma completa ed innovativa per dispositivi mobili rilasciata con licenza open-source ad Ottobre 2008.

Il concetto importante che sta alla base dello sviluppo di Android è riassunto dalla parola **Open**. Android è aperto ad ogni livello della sua architettura, utilizza Linux come Base System OS per una maggiore portabilità, il framework di librerie è costituito da un mix tra progetti open-source esterni e componenti interni rilasciati anche questi con licenza open, infine le applicazioni vengono sviluppate in un modello collaborativo mai visto prima: ogni applicazione è libera di utilizzare tutte le funzionalità presenti nella piattaforma senza differenze tra applicazioni base di Android e quelle prodotte da singoli sviluppatori. Inoltre ogni applicazione è formata da componenti che altre applicazioni possono riutilizzare o sostituire per abbattere i tempi di sviluppo ed evolvere le funzionalità presenti liberamente.

Questo capitolo si basa su documentazione ricavata dagli eventi Google I/O [5, 16] ed altre informazioni ricavate nella rete.

2.1 Licenza della piattaforma

La licenza di riferimento del progetto open-source Android è Apache 2.0; licenza free software approvata dalla comunità Open Source Initiative

[1] `http://www.openhandsetalliance.com/`

(OSI).

Come una qualsiasi licenza free software, la licenza Apache garantisce agli utilizzatori del software la libertà di utilizzarlo, distribuirlo, modificarlo e distribuirne prodotti derivati. Apache 2.0 però non è copyleft: non esprime obblighi rispetto al regime giuridico dei prodotti derivati i quali, al limite, possono essere distribuiti con licenza proprietaria. E' imposto però l'obbligo di citazione del lavoro originale.

Ciò che ha spinto Google ad utilizzare questa tipologia di licenza è proprio la libertà rispetto alla forma di licenza dei lavori derivati: Per sua natura Android è una piattaforma espandibile e personalizzabile, aziende produttrici di dispositivi mobili possono si contribuirne allo sviluppo ma, in questo modo, possono anche inserire o modificare componenti (User Interface ..) senza dover necessariamente rilasciare il codice sorgente con licenza open-source.

2.2 Architettura

Android si presenta con una architettura multi-layer a componenti mostrata in figura 2.1 ed analizzata qui nel dettaglio.

2.2.1 Kernel Level

Come scritto in precedenza Android è basato sul sistema operativo Linux 2.6 con alcune estensioni (per questo lo stack Android è detto *Linux-derived* invece di *Linux-based*).

Linux fornisce una solida base alla piattaforma Android: fornisce modelli consolidati nella gestione dei *processi*, della *memoria*, dei *driver* di periferica, supporta *librerie condivise* ed implementa un ottimo modello di *sicurezza* basata su permessi. Inoltre supporta una grande quantità di piattaforme hardware consolidate ed è facilmente portabile in quelle ancora in sviluppo disponendo di una grossa comunità di sviluppatori continuamente attiva nell'evolvere il sistema operativo per mantenerlo al passo con i requisiti delle moderne tecnologie.

Linux però è sviluppato principalmente per servire sistemi informatici con capacità computazionali elevate e senza particolari limiti nel consumo di energia, fattore limitante per il target dei dispositivi mobili. Per migliorare la gestione dell'energia e per implementare le funzionalità criti-

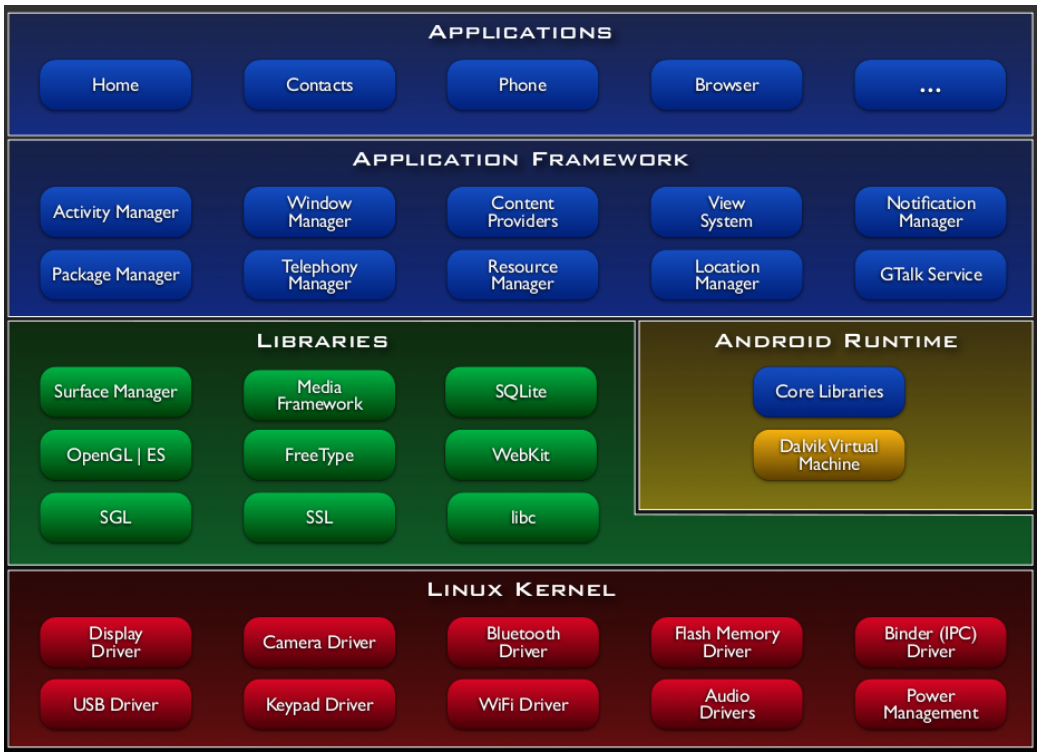


Figura 2.1: Stack di Android.

che del modello delle applicazioni Android, sono stati introdotti nel kernel i moduli seguenti:

IPC Binder

Il modulo Binder fornisce un metodo evoluto per il meccanismo Inter Process Communication (IPC) simile agli standard *CORBA* e *Java RMI* per Sistemi Distribuiti.

Nasce dallo sviluppo di un vecchio progetto interno a PalmSource: *OpenBinder*, pubblicato con licenza open source nel 2005 e del quale esiste ancora la documentazione online[2].

Questa tipologia di architettura IPC permette interazioni complesse tra processi difficilmente riproducibili attraverso meccanismi standard *POSIX* se non con un elevato overhead computazionale: ogni processo, più

[2] `http://www.angryredplanet.com/~hackbod/openbinder/`

precisamente ogni thread, viene considerato un entità capace di esporre o utilizzare "servizi" di altre entità.

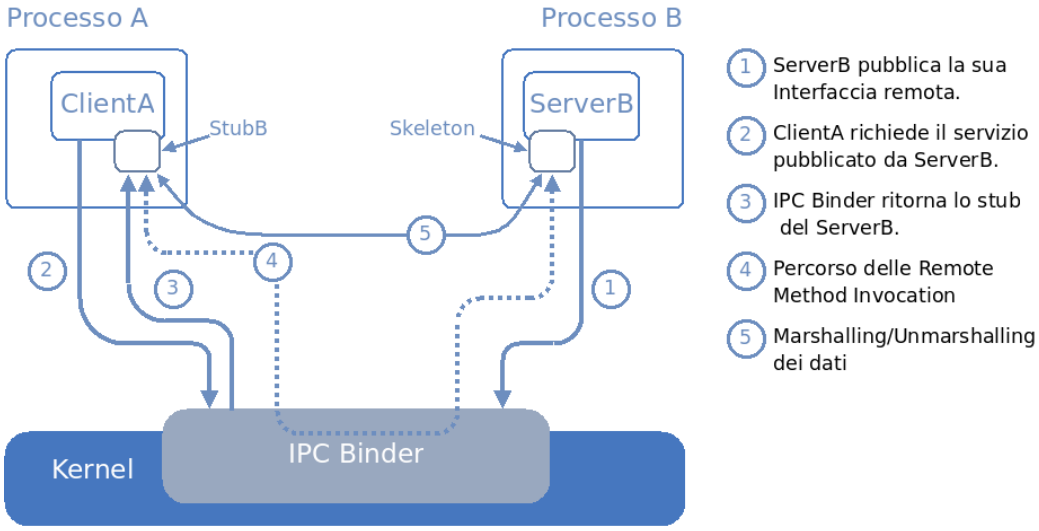


Figura 2.2: Modello di IPC introdotto da Android Binder.

Essenzialmente è un driver in kernel più librerie user space capaci di gestire l'intero meccanismo IPC tra i processi in esecuzione nel sistema riassunto nella figura 2.2 quindi: l'iscrizione la ricerca e l'acquisizione dei servizi disponibili pubblicati dai componenti Services attivi secondo il meccanismo publish/subscribe, il meccanismo di marshalling/unmarshalling dei dati, la comunicazione vera e propria attraverso memoria condivisa e la gestione della consistenza delle referenze "remote" create dall'acquisizione dei servizi.

La definizione dell'interfaccia pubblica avviene tramite il linguaggio Android Interface Definition Language (AIDL) capace di astrarre i diversi linguaggi di programmazione con i quali vengono realizzati i servizi in un paradigma ad oggetti.

Dato che deve essere utilizzato prevalentemente in sistemi embedded, Binder è stato ottimizzato nella dimensione dei messaggi e nella quantità di process-switch necessari.

Low Memory Killer

Modulo in grado di terminare processi nel caso sia rimasta poca memoria disponibile.

Accetta in ingresso due array `adj` e `minfree` e sfrutta il parametro `oom_adj` assegnato ad ogni processo. Finché la memoria libera rimane al di sotto di `minfree[i]` pagine, processi con valori `oom_adj` maggiori o uguali a `adj[i]` vengono terminati.

A differenza del componente Out Of Memory (OOM) killer di Linux (che agisce terminando innanzitutto i processi più avidi di risorse) il nuovo modulo Android permette algoritmi personalizzati per gestire la mancanza di memoria attraverso interventi al valore `oom_adj` per ogni processo a seconda dell'importanza attuale all'interno dell'ambiente d'esecuzione.

Ashmem

Anonymous SHared MEMory system (Ashmem) definisce un interfaccia attraverso la quale regioni di memoria possono essere condivise tra processi attraverso un nome.

Utilizzata per ottimizzare il consumo di memoria grazie a pool di risorse comuni (icone ..) a differenza del meccanismo di condivisione della memoria shmem di Linux, permette al kernel di liberare la regione condivisa se questa non è correntemente in uso anche se permangono referenze valide alla stessa regione (shmem libera il blocco condiviso solo se sono stati liberati da tutti gli handle a tale regione).

Un processo che tenta di accedere a memoria condivisa liberata dal kernel riceverà un errore e se necessario dovrà ri-allocare il blocco e ricaricarne i dati.

Power Management

La gestione dell'energia è cruciale in un dispositivo mobile. Android inserisce un layer software al di sopra del componente Power Management di Linux per aumentarne l'aggressività: vengono definiti quattro stati di carico possibili nel sistema:

Stato	**Descrizione**
Full power:	CPU e LCD attivi.
LCD Off:	solo la CPU attiva è attiva.
Energy save:	CPU e LCD in massimo risparmio energetico.
Off:	Il sistema è spento.

Le transizioni da uno stato all'altro dipendono dagli input dell'utente e possono essere regolate via software: ogni processo (unità di esecuzione) può richiedere di bloccare il sistema in un particolare stato tramite l'acquisizione di oggetti `wake_lock` che possono essere principalmente di due tipi:

FULL wake_lock: il sistema verrà bloccato nella modalità *Full power* finché ogni *wake_lock* di questo tipo verrà rilasciato.

PARTIAL wake_lock: la CPU è forzata a rimanere attiva (gli stati possibili sono *Full power* e *LCD Off*); viene utilizzato da applicazioni che non richiedono esplicitamente il display ed interpretano la volontà dell'utente di eseguire in modo continuativo fino a nuovo ordine (mp3 player ..).

Questo intervento non preclude altri meccanismi di gestione dell'energia, per questo è fortemente consigliato il supporto al frequency scaling della CPU se presente, per adattarne il consumo alle necessità computazionali dei due stati che la vogliono attiva.

RAM Console, Log Device, Android Debug Bridge

Moduli di utilità diagnostica:

RAM Console: permette di salvare i messaggi di log del kernel in un buffer in memoria.

Logger: implementa un sistema di logging unificato per tutte le entità user-space, al quale può connettersi il modulo *RAM Console*.

Android Debug Bridge (ADB): implementa un canale di comunicazione attraverso bus USB tra il sistema target (che esegue Android) e la workstation di sviluppo tramite il quale è possibile: installare applicazioni, scambiare file, ottenere le stringhe di log del sistema ed aprire una shell remota di comando.

Alarm

Modulo creato per gestire a livello kernel la funzionalità di sveglia. Se nella piattaforma hardware è presente un orologio di sistema Real Time Clock (RTC) correttamente configurato, questo potrà essere istruito attraverso il modulo Alarm per riattivare il sistema ad un'ora stabilita.

2.2.2 Framework Libraries

Scritte in C/C++, forniscono le funzionalità di base necessarie agli strati superiori ed implementano quei servizi per i quali le performance d'esecuzione sono un fattore critico del sistema.

Bionic Libc

Implementazione personalizzata delle librerie C/C++ ottimizzata per l'uso nei sistemi embedded.

Le librerie Bionic sono state sviluppate per essere leggere e veloci implementando le sole funzionalità ritenute necessarie dello standard POSIX: non esiste il supporto per le *eccezioni* in C++, per i caratteri *wchar* (concetto già ampiamente sostituito dallo standard unicode) e l'implementazione della libreria *pthread* si mostra alleggerita di funzionalità non essenziali.

Oltre a voler essere uno strumento agile le librerie C/C++ in Bionic libc sono state adattate al sistema operativo Android implementando funzionalità specifiche come l'interfaccia verso il sistema di log e verso il database delle proprietà.

Oltre alla questione prestazionale e di personalizzazione delle funzionalità delle librerie C/C++, ciò che ha spinto gli sviluppatori di Android per una re-implementazione dalla base è stata anche la questione della licenza di distribuzione: dato che le librerie C vengono collegate o linkate staticamente ad ogni applicazione sviluppata per Android, componenti sviluppati con licenza proprietaria non potrebbero essere inseriti nel sistema se la licenza delle librerie fosse GPL o GPL-derived come nel caso delle uClibc.

Per questo Bionic Libc è rilasciata con licenza BSD la quale permette l'inserimento di codice BSD in progetti rilasciati con licenza proprietaria (anche closed-source) purché venga esplicitato il copyright originale.

Maggiori informazioni nell'articolo [35] e nel documento non ufficiale [4].

Surface Flinger

Server che gestisce la fusione dei differenti layer grafici (`Surface`) verso l'immagine prodotta a schermo. Gli oggetti `Surface` possono essere di dimensioni variabili e contenere grafica 2D o 3D, possono essere creati

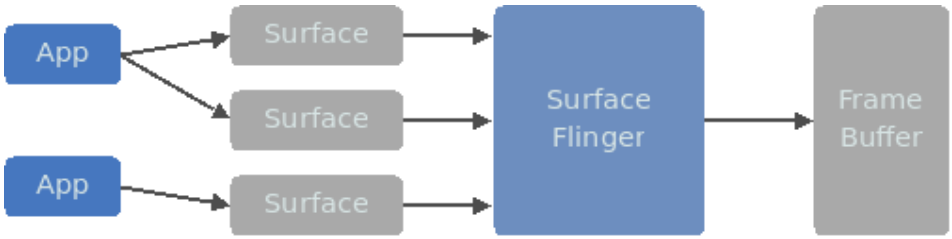


Figura 2.3: Workflow del server Surface Flinger.

indirettamente da un'applicazione (tramite oggetti grafici) oppure da un server multimediale. In Surface Flinger il rendering può avvalersi delle librerie *OpenGL ES* in campo 3D ed accelerazioni hardware se presenti per grafica 2D.

Per aumentare la fluidità delle animazioni e limitare possibili errori di rendering, in Surface Flinger è implementata una sorta di *double buffering*: se il driver video supporta uno schermo virtuale di dimensioni doppie rispetto all'immagine visualizzata, allora Surface Flinger è programmato per scrivere i nuovi frame alternando nelle porzioni dello schermo virtuale non visualizzate. Solo al completamento del nuovo frame, lo schermo virtuale verrà traslato nella corretta posizione eliminando così visualizzazioni di frame incompleti.

Audio Flinger

Come Surface Flinger, il server Audio Flinger si occupa di multiplexare i differenti canali audio provenienti dai diversi Server (*Tone Audio*, *Media Player*, ..). Gestisce inoltre il routing del flusso ottenuto su i diversi dispositivi di output (*Cuffie*, *Altoparlante*, *Bluetooth* ..).

SQLite, WebKit, Media Framework

Progetti Open-source di supporto alle attività di base inclusi nella piattaforma:

SQLite [3] - Relational Database Management System (RDBMS) molto leggero utilizzato per memorizzare ed organizzare gran parte dei dati della piattaforma in un database relazionale.

[3] `http://www.sqlite.org/`

WebKit [4] - Motore di rendering per pagine web capace di sfruttare appieno le dimensione del display di un dispositivo mobile attraverso la visualizzazione full desktop view (senza barre degli strumenti) e la possibilità di navigare dinamicamente all'interno della pagina con semplici pan e zoom.
Implementa tecnologie a supporto del Web 2.0 quali *CSS*, *Javascript*, *DOM*, *AJAX*. Viene utilizzato anche dalla Apple nel suo browser Safari e nella piattaforma S60.

OpenCORE [5] - Sviluppato da PacketVideo questo Media Framework fornisce la capacità di riprodurre e/o registrare flussi video da file o stream di rete. Sono supportati i più diffusi codec audio/video inclusi quelli sviluppati appositamente per il mondo mobile tra cui *3GPP*, *MPEG-4*, *AAC*, *MP3*, *H.263*, *H.264* e fornisce un'interfaccia a plug-in per il supporto di quelli mancanti.
Questo modello di gestione dei codec permette di sviluppare plug-in specifici per i singoli dispositivi capaci di supportare le singole accelerazioni hardware multimediali se disponibili.
OpenCORE presenta inoltre uno strato Content Policy Manager per gestire i contenuti protetti da copyright[6] necessaria per supportare applicazioni che accedono ai servizi globali di distribuzione multimediale in rete.

Dalvik

Rappresenta una delle maggiori innovazioni di Android: Dalvik è una nuova implementazione di *Java Virtual Machine* ottimizzata per dispositivi mobili sulla quale si basa tutto l'ambiente applicativo.

Dalvik è il punto di collegamento tra la semplicità dello sviluppo di applicazioni in linguaggio Java e lo sfruttamento dei modelli di sicurezza e gestione dei processi offerti da Linux con particolare attenzione alle prestazioni in termini di spazio occupato dal file eseguibile, gestione della memoria e velocità d'esecuzione:

Sicurezza - Ogni applicazione in Android corrisponde ad un processo Linux che esegue in seno ad una propria istanza Dalvik privata. Questo permette di garantire l'isolamento della memoria a disposi-

[4] `http://webkit.org/`
[5] `http://www.pv.com/products/android/index.html`
[6] `http://www.androidguys.com/2008/06/03/34-weeks-of-oha-10-2/`

zione della singola applicazione e di costruire politiche di gestione degli accessi sfruttando i concetti di utenti e gruppi Linux.

Gestione della Memoria - La moltiplicazione di istanze JVM così ottenuta ha richiesto ottimizzazioni stringenti nella gestione della memoria rispetto alle JVM standard. Per questo è stato sviluppato un nuovo formato di file eseguibile *.dex (Dalvik EXecutable)* capace di contenere più definizioni di classi (alla pari dei file .jar che contengono più file .class) riordinate in modo da eliminare la ripetizione di aree funzionali e dati comuni ottenendo così una riduzione di più del 50% dello spazio occupato dai componenti di Android (Core Libraries, Application Framework ..) in un formato eseguibile non compresso.

Prestazioni - Sforzi sono stati fatti per migliorare le prestazioni dell'interprete attraverso ottimizzazioni in fase di installazione (static linking, "inlining" di speciali metodi, pruning di metodi vuoti ..) e definendo un nuovo set di istruzioni più ricco per limitare il numero di instruction dispatch e letture/scritture accessorie in memoria.

A supporto della JVM Dalvik sono state implementate le librerie *Android Core Libraries* per ricreare il sottoinsieme utile delle API fondamentali Java (strutture dati, accesso ai file ..) e fornirne di nuove specifiche per l'interfacciamento con le librerie ed i servizi degli strati inferiori (Grafica, Network, Web ..).

Per maggiori informazioni su Dalvik si faccia riferimento al documento [40].

Hardware Abstraction Layer

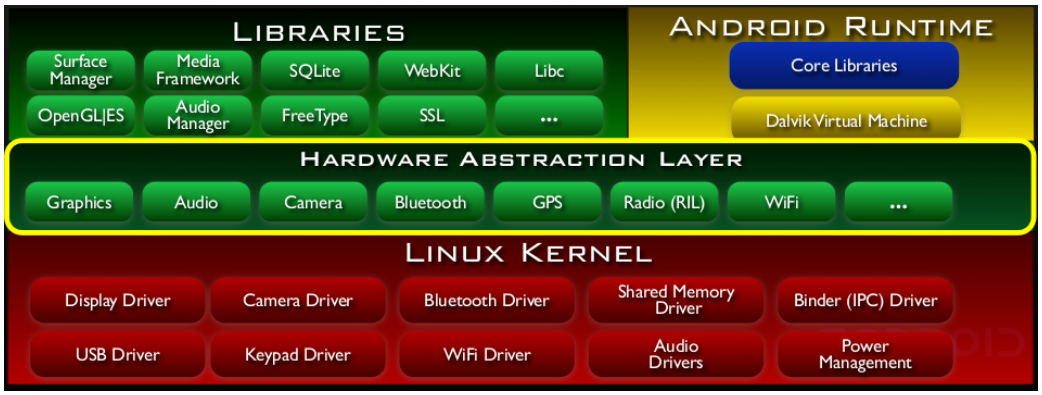


Figura 2.4: Posizionamento del' Hardware Abstraction Layer Android.

Android inserisce un layer di astrazione tra librerie di sistema ed il sistema operativo Linux per unificare casi specifici di comportamenti driver non standard e fornire un livello di separazione tra componenti Android e software GPL.

L'intero layer è sviluppato in C/C++ e definisce, per ogni tipologia di dispositivo, un'interfaccia di controllo da implementare in uno "stub" di comunicazione verso il driver Linux vero e proprio oppure, in genere, verso l'interfaccia esposta da una famiglia di driver (es. per il componente Camera sono possibili le estensioni verso le interfacce v4l, v4l2 ..).

2.2.3 Application Framework

Strato in blu dello stack presentato in figura 2.1. Comprende tutti quei server in esecuzione nell'ambiente Android sviluppati alla pari delle applicazioni che gestiscono attivamente funzionalità del sistema operativo:

Activity Manager: Gestisce il ciclo di vita delle applicazioni e mantiene ordine nell'insieme di quelle aperte attraverso stack dove è possibile navigare nel caso si volesse tornare indietro premendo il tasto Back (es: Dalla Home apro E-mail client per leggere le mail, aperta una Mail voglio visualizzare il documento pdf allegato. Tornare indietro dall'applicazione Pdf viewer significa ripercorrere all'indietro lo stack creato ritornando alla visualizzazione della Mail precedente).

Package Manager: Utilizzato dal server Activity Manager per caricare le informazioni esposte dalle applicazioni nei propri file Manifest interni ai pacchetti .apk. Mantiene conoscenza di tutte le applicazioni caricate e delle capacità pubblicate dalle stesse.

Window Manager: Server che gestisce lo schermo. Dato l'insieme di applicazioni in esecuzione ne determina l'ordine di visualizzazione da comunicare al server Surface Flinger.

Resource Manager: Manager delle risorse di sistema. Fornisce l'interfaccia per acquisire risorse come stringhe, icone, file video, audio, proprie di Android riducendo gli sprechi di memoria attraverso un fruizione il più possibile condivisa.

Content Provider: Simile al Resource Manager, fornisce un metodo unificato per accedere ai contenuti, intesi come dati (es contatti del-

la rubrica) o file multimediali memorizzati nel database di sistema, accessibili localmente o via rete.

View System: Gestisce l'assemblaggio dei componenti grafici che andranno a creare l'interfaccia delle singole applicazioni.

Notification Manager: Gestisce la presentazione all'utente degli eventi. Questi possono essere eventi di sistema (Low battery, ...) oppure eventi creati appositamente da un'applicazione (messaggio in arrivo, appuntamento..). La notifica viene presentata nella barra delle notifiche e può avvalersi di suoni, vibrazioni e grafica.

Inoltre sono presenti dei server di interfaccia verso i dispositivi hardware disponibili che sono: *Telephony Service*, *Location Service* (GPS), *Bluetooth Service*, *WiFi Service*, *USB Service*, *Sensor Service* (accelerometri, compasso ..). E' grazie alle interfacce esposte da questi che le applicazioni possono accedere ai driver di periferica attraverso lo stack di sistema.

Modello delle Applicazioni[36]

In Android le applicazioni sono scritte in linguaggio Java; il bytecode compilato e le risorse necessarie alla singola applicazione vengono riuniti con l'ausilio del tool **aapt** in un pacchetto *.apk* (Android Package). E' questa la forma in cui una applicazione può essere distribuita ed installata in un dispositivo reale.

Ogni applicazione in esecuzione nel sistema operativo, se lanciata, esegue in un proprio processo Linux in seno a una propria istanza JVM privata; lo *User ID* del processo viene assegnato da Android in modo univoco nell'insieme dei processi in esecuzione (se non esplicitamente definito) e i permessi dei file vengono impostati così che ogni applicazione non possa accedere direttamente a risorse non proprie.

Componenti di una applicazione

Una caratteristica fondamentale del modello applicativo Android è la possibilità di rompere i confini delle singole applicazioni considerando queste un'insieme di componenti riutilizzabili o sostituibili. Il vantaggio finale è la velocità di sviluppo: chi scrive codice per Android non deve reimplementare tutto da capo, ma può avvalersi di componenti già presenti e disponibili nel sistema richiamandoli al momento giusto. Per questo una applicazione non contiene un main entry point (metodo `main()`) ma

è costituita da un'insieme di componenti collegati tra di loro sviluppati per eseguire un determinato compito.

Esistono quattro tipologie di componenti:

Activities - Una attività presenta un'interfaccia all'utente da disegnare in una singola finestra per uno scopo specifico come visualizzare un gruppo di immagini, scegliere un'opzione, visualizzare la lista delle mail ricevute.
Android, al momento del lancio dell'applicazione, esegue l'attività marcata come principale e questa, a seconda degli input dell'utente, attiverà le attività successive formando così l'intera interfaccia utente dell'applicazione.

Services - Un servizio è un contenitore per compiti da eseguire in background senza la necessità di una interfaccia utente. Viene utilizzato per mantenere in esecuzione determinati compiti (come la riproduzione di un file audio) che l'utente si aspetterebbe continuassero anche navigando in applicazioni differenti.
Come ogni componente di una applicazione anche i servizi eseguono all'interno del main thread nel processo assegnato; ciò impone per compiti di lunga durata di eseguirli in thread separati per non bloccare gli altri componenti dell'interfaccia utente. I servizi possono esporre una interfaccia remota definita tramite il linguaggio AIDL per potervi accedere tramite IPC.

Broadcast receivers - Componenti in genere dormienti che ricevono e reagiscono a eventi inviati a tutte le applicazioni. Gli eventi possono essere propri del sistema (come low battery, cambiamento della time-zone ..) oppure generati da altre applicazioni (una applicazione che comunica alle altre d'aver completato il download di una risorsa e quindi è pronta all'uso).

Content Providers - Utilizzati per mettere a disposizione un specifico sottoinsieme di dati della propria applicazione ad altre applicazioni.

Nel momento in cui venga richiesto uno specifico componente di una applicazione, Android fa si che il processo legato a questa sia in esecuzione, iniziandolo se necessario.

2.2.4 Il file Manifest

Ciò che da ragione al contenuto di un pacchetto .apk è il file *AndroidManifest.xml* al suo interno. Il file Manifest descrive tramite codice XML

tutto ciò che contiene il pacchetto .apk:

- Le risorse: immagini, audio, video, stringhe.
- I vari componenti dell'applicazione riuniti per tipologia. Nella lista delle attività deve essere esplicitata quella principale ed in genere, per ogni componente vengono esplicitati gli *intent-filter*: metadati che identificano le capacità esposte dal componente specifico così che un'applicazione esterna possa accedervi attraverso una ricerca per capacità.
- I permessi che l'applicazione si aspetta gli vengano assegnati: di accesso ai specifici servizi (tramite ID di gruppo), condivisione di risorse con applicazioni specifiche, etc...
- Le librerie oltre alle Core Libraries necessarie all'applicazione.

Per una descrizione approfondita dei file Manifest di Android si faccia riferimento alla documentazione online [37].

2.2.5 Attività e componenti affini

Le attività nell'ambiente di esecuzione Dalvik vengono individuate comunicando al server Activity Manager oggetti di tipo *Intent*.

Ciò significa che dall'attività principale, per lanciare un'attività accessoria deve essere creato un oggetto che la descriva, attraverso il suo nome univoco se l'obbiettivo è identificare un componente specifico, oppure attraverso una chiave di ricerca se si vuole utilizzare un componente generico (presente nella stessa applicazione oppure parte di applicazioni esterne).

Nel caso di richiesta generica la chiave di ricerca verrà confrontata con tutti gli *intent-filter* pubblicati per ogni componente del sistema alla ricerca del matching migliore, attivando così la relativa attività.

2.2.6 Ciclo di vita dei componenti

Activity

Dalla creazione alla distruzione, ogni attività segue il proprio ciclo di vita all'interno del diagramma stati-transizioni mostrato in figura2.5.

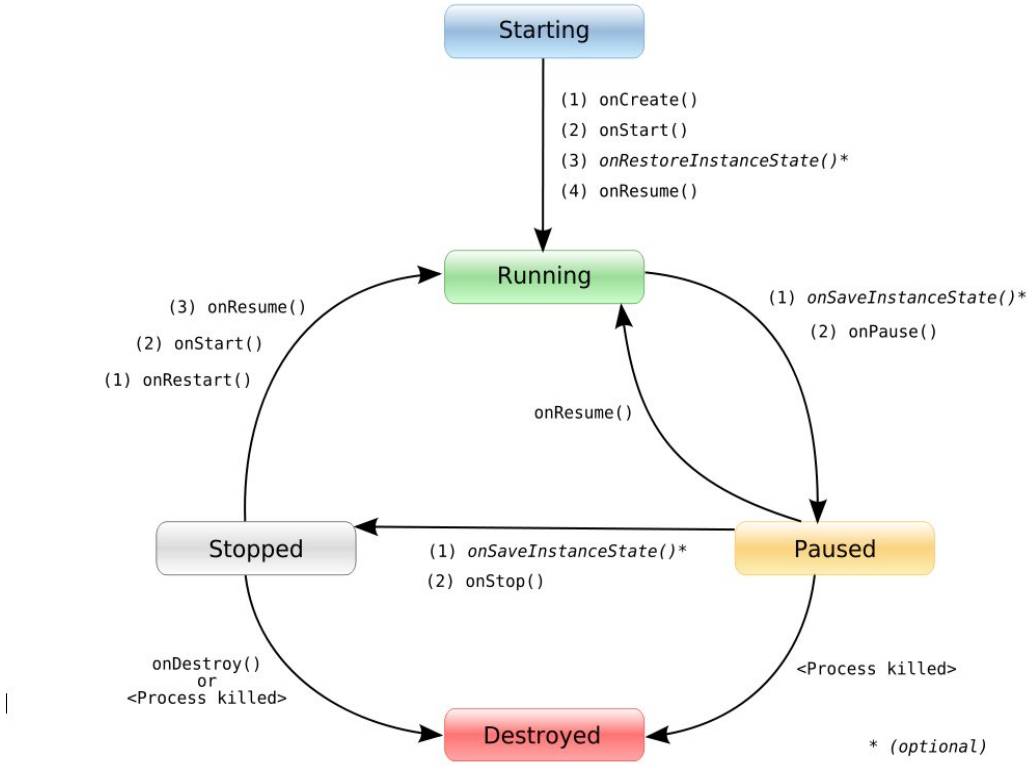


Figura 2.5: Ciclo di vita del componente Activity.

Essenzialmente una attività può risiedere in uno dei tre stati:

Running - è in primo piano nello schermo, l'utente può interagire con questa attività.
Una attività in questo stato non può essere distrutta da Android.

Paused - ha perso il fuoco ma è ancora visibile all'utente. Accade quando un'altra attività viene mostrata sopra la prima con una finestra trasparente o che non copre l'intero schermo.
Una attività in pausa può essere distrutta solamente in caso di estrema mancanza di memoria.

Stopped - è stata completamente oscurata da un'altra attività .
In caso di mancanza di memoria, sono innanzitutto queste le attività che vengono sacrificate per prime.

Activity Manager permette alle attività di avere coscienza delle proprie transizioni di stato chiamando i metodi seguenti nell'ordine mostrato in figura 2.5.

`void onCreate(Bundle savedInstanceState)` Metodo che ha il compito di inizializzare tutti gli oggetti necessari all'attività .
L'oggetto `savedInstanceState` è lo stesso che riceverebbe la funzione **`onRestoreInstanceState()`** nel processo di riattivazione della attività.

`void onStart()` Chiamato subito prima della visualizzazione dell'attività , seguito da **`onResume()`** se la transazione può procedere oppure da **`onStop()`** in caso di problemi.
Assieme a **`onStop()`** delimita la porzione visibile del ciclo di vita dell'attività , per questo dovrà essere utilizzato per creare ed acquisire tutte quelle risorse necessarie alla visualizzazione della stessa, come la creazione di Broadcast Receiver che avranno impatti nella User Interface.

`void onRestart()` Chiamato all'inizio della transazione dallo stato Stopped allo stato Running.

`void onResume()` Dopo la visualizzazione a schermo dell'attività , precede l'inizio dell'interazione con l'utente.
Delimita assieme ad **`onPause()`** la porzione attiva del ciclo di vita dell'attività .

`void onPause()` Precede la perdita del controllo dello schermo nel caso un'altra attività lo richieda.
Dato che questo è l'unico metodo di cui vi è la certezza dell'esecuzione prima che il componente diventi vulnerabile al meccanismo di recupero della memoria, allora è qui che una attività ha il dovere di completare le transazioni sui dati persistenti controllati. Ciò non riguarda lo stato dell'interfaccia grafica (posizione attuale all'interno della lista, valori degli input specifici dell'istanza attuale ..) considerato volatile tra due esecuzioni differenti dell'applicazione, salvato tramite oggetti `Bundle` nel metodo opzionale **`onSaveInstanceState()`**.

`void onStop()` Chiamato durante il processo di distruzione dell'attività dopo che questa non è più visibile.
Ha il compito di rilasciare le risorse volatili acquisite in **`onStart()`** non necessarie negli istanti in cui l'attività non è visualizzata.

`void onDestroy()` Metodo che termina il processo normale di distruzione di una attività.

Ha il compito di rilasciare le risorse create in **`onCreate()`** compreso terminare i thread accessori ancora in esecuzione.

`onSaveInstanceState()` e `onRestoreInstanceState()` Metodi opzionali chiamati solamente nel caso le transazioni dell'attività avvengono per cause dipendenti dal sistema operativo e non per azioni specifiche dell'utente.
In questo caso l'attività può trovarsi negli stati Paused o Stopped vulnerabile alla terminazione da parte di Android in mancanza di memoria disponibile. L'utente non ha voluto chiudere l'applicazione per cui si aspetta che, riportandola nello stato Running (navigando nello stack delle attività aperte), presenti lo stesso stato volatile dell'interfaccia grafica lasciata in precedenza.
Allora **`onSaveInstanceState()`** ha il compito di salvare lo stato volatile dell'attività in un oggetto di tipo `Bundle` (sequenze di chiavi e valori) affidato al server Activity Manager che provvederà a passarlo come parametro del metodo **`onRestoreInstanceState()`** chiamato nel processo di riattivazione dell'attività dopo che questa è stata distrutta.

Service

Come le attività anche i servizi possiedono metodi che ne caratterizzano la posizione all'interno del loro ciclo di vita. In Android esistono due modalità per usufruire di un servizio a seconda della complessità di interazione voluta:

Semplice : Nel caso non sia necessario interagire con il servizio tramite metodi di interfaccia oppure il servizio non ne implementa affatto una.
In questo caso un componente può creare il servizio con **`Context.startService(Intent service)`**. Activity Manager provvederà a chiamare in sequenza le callback **`onCreate()`** e **`onStart()`** del servizio con i rispettivi compiti di inizializzazione delle risorse necessarie ed inizializzazione delle attività da eseguire. Ripetute chiamate a **`startService()`** sullo stesso servizio si produrranno in ripetute chiamate a **`onStart()`** fornendo così un metodo allo stesso servizio per lavorare su più risorse.
Un servizio può essere terminato tramite una chiamata a **`Context.stopService(Intent service)`** (non importa quante siano state le chiamate a **`startService()`**) oppure dall'interno con

il metodo **`stopSelf()`**. Queste azioni produrranno la chiamata alla callback **`onDestroy()`** da parte di Activity Manager, il cui compito è fermare tutte le attività in corso (thread accessori) e liberare le risorse del servizio.

Remota : Nel caso sia necessario interagire con il servizio creato attraverso la sua interfaccia pubblica.
In questo caso un componente client può accedere al servizio con **`Context.bindService()`**. Activity Manager provvederà alla ricerca del servizio voluto tra quelli già attivi (iniziati con chiamate a **`bindService()`** oppure a **`startService()`**) creandolo se necessario.
La callback chiamata in seguito all'operazione di bind è **`onBind()`** (preceduta da **`onCreate()`** nel caso di nuova attivazione) terminata la quale il servizio sarà pronto all'interazione tramite interfaccia remota.

Il client potrà disconnettersi dal servizio tramite il metodo**`Context.unbindService()`** producendo la chiamata alla callback **`onUnbind()`** il cui compito è liberare le risorse utilizzate nella comunicazione.
`onUnbind()` può richiedere di far sopravvivere il servizio in uno stato dormiente, anche se non ci sono connessioni attive, in questo caso nuove richieste di connessioni si produrranno in chiamate alla callback **`onRebind()`**.

Per Android un Servizio è non interrompibile se è utilizzato dall'attività in primo piano nel sistema oppure quando sono in esecuzione una delle sue callback che marcano il ciclo di vita (**`onCreate()`**, **`onStart()`**, **`onDestroy()`**, **`onBind()`**, **`onUnbind()`**, **`onRebind()`**).

Broadcast Receiver life-cycle

Un Broadcast Receiver possiede una sola callback: **`onReceive (Context, intent)`** alla quale è affidato il compito di reazione all'evento configurato. L'intervallo di esecuzione di questa callback viene considerato come il solo non interrompibile da Android. Per tutto il resto del tempo questo componente, se risiede in un processo non attivo, può essere terminato per liberare memoria.

Dato che l'esecuzione della callback deve essere il più possibile veloce (altrimenti il main thread del processo nel quale risiede ne viene bloccato)

e che thread creati da questo componente non sono considerati attività protette da Android, elaborazioni di risposta ad eventi che richiedono tempo dovrebbero essere eseguite da un servizio accessorio.

2.2.7 Processi e ciclo di vita dei componenti

Nel caso l'insieme di applicazioni aperte in un dato istante produca una situazione critica nel sistema in termini di risorse di memoria occupate, Android provvede a terminare processi considerati di minore utilità.

In Android i processi vengono ordinati in una scala gerarchica a seconda dei componenti che contengono e della posizione che hanno nel loro ciclo di vita. Allora i processi che stanno al livello inferiore sono quelli che prima di tutti verranno terminati nel caso il livello di risorse disponibili sia molto basso via via così fino ai processi che stanno in cima nella scala gerarchica che subiranno l'intervento del sistema solamente nel caso non siano disponibili i requisiti minimi di risorse per tale applicazione.

Sono definiti cinque livelli nella gerarchia rappresentati in seguito in ordine di importanza:

1. Un **processo attivo** è un processo in seno all'applicazione correntemente utilizzata dall'utente.
   Un processo è attivo se:
   - Sta eseguendo l'attività con la quale l'utente sta interagendo.
   - Contiene un servizio al quale l'attività utilizzata correntemente dall'utente è connessa attraverso bind.
   - Contiene un servizio che sta eseguendo una delle sue life-cycle callback.
   - Contiene un Broadcast Receiver che sta eseguendo la propria callback `onReceive()`.
2. Un **processo visibile** è un processo che non possiede nessun componente attivo ma che comunque influenza lo schermo dell'utente. Processi di questo tipo sono considerati molto importanti.
   Un processo è visibile se:
   - Contiene un'attività visibile in secondo piano (nello stato Paused).
   - Contiene un servizio collegato ad una attività visibile.
3. Un **processo di servizio** è un processo che contiene almeno un servizio attivato con il metodo `startService()` che non ricade nelle

due gerarchie superiori.
I servizi "stand-alone" di questo tipo sono considerati importanti perché possono eseguire attività importanti ma non vitali per l'utente come riprodurre un file musicale.

4. Un **processo in background** è un processo che non influenza l'interfaccia utente ne le elaborazioni in corso.
Navigando nell'ambiente Android un buon numero di processi creati possono ricadere in questa categoria, questi vengono ordinati in una coda LRU (least recently used) e terminati a partire dal più vecchio per essere ricaricati nel caso l'utente ne richieda la visualizzazione. Se le callback delle attività sono implementate correttamente questo meccanismo avviene in modo trasparente all'utente.

5. Un **processo vuoto** è un processo che non contiene nessun componente attivo.
Il motivo che spinge a mantenere un processo del genere è l'effetto cache ottenuto nel caso l'utente ri-acceda ad una applicazione recentemente chiusa.

2.3 Android SDK

Il primo Software Development Kit (SDK) Android è stato rilasciato in Agosto 2008. Si tratta di una serie di tool per lo sviluppo e test di applicazioni Android, disponibile per tutte le maggiori piattaforme PC (Linux Windows e MAC) nel sito ufficiale[7].

L'SDK contiene:

Un emulatore di handset Android: viene utilizzato Qemu[8] per creare un ambiente di emulazione dell'architettura ARMv5.
Vengono fornite quindi le immagini del kernel sviluppato appositamente (versioni *goldfish*), ed un root-filesystem contenente lo stack Android.

Librerie di sviluppo: nel pacchetto sono presenti le librerie Android complete in formato *.jar*. In questo modo è possibile compilare una applicazione con l'SDK Java fornito da *sun*. I binari verranno poi tradotti nel formato *.dex* e poi uniti nel pacchetto

[7] `http://developer.android.com/sdk/`
[8] `http://www.nongnu.org/qemu/`

.apk per l'installazione e l'esecuzione nell'emulatore. Una applicazione sviluppata tramite SDK può essere eseguita in un qualsiasi dispositivo reale.

Debugger: L'SDK contiene l'utility Android Debug Bridge (*adb*).

Documentazione, Esempi e Tutorial: Tutte le informazioni necessarie per iniziare a sviluppare applicazioni per Android.

Il modo più semplice di utilizzare l'SDK Android è attraverso il Plugin Eclipse acfADT scaricabile dalla stessa posizione indicata per l'Android Software Development Kit.

2.3.1 Personalizzare le versione di Android eseguita dall'emulatore

Successivamente l'installazione dell'Android Software Development Kit, nella workstation si dispone di Qemu: un emulatore generico di architettura ARMv5 personalizzabile in molti aspetti.

Eseguendo da una shell:

```
$ emulator -help
Android Emulator usage: emulator [options] [-qemu args]
  options:
    -system <dir>            read system image from <dir>
    -datadir <dir>           write user data into <dir>
    -kernel <file>           use specific emulated kernel
    -ramdisk <file>          ramdisk image (default <system>/
       ramdisk.img
    -image <file>            system image (default <system>/
       system.img
    -initdata <file>         initial data image (default <system
       >/userdata.img
    -data <file>             data image (default <datadir>/
       userdata-qemu.img
...
    -sdcard <file>           SD card image (default <system>/
       sdcard.img
    -wipe-data               reset the use data image (copy it
       from initdata)
...
    -show-kernel             display kernel messages
    -shell                   enable root shell on current
       terminal
```

```
    -nojni                    disable JNI checks in the Dalvik
       runtime
    -logcat <tags>            enable logcat output with given
       tags
...
    -dns-server <servers>     use this DNS server(s) in the
       emulated system
    -cpu-delay <cpudelay>     throttle CPU emulation
    -no-boot-anim             disable animation for faster boot
    -no-window                disable graphical window display
    -version                  display emulator version number
    -report-console <socket> report console port to remote
       socket
...
```

E' possibile verificare quali sono le opzioni che permettono di personalizzare le versioni del kernel e dello stack Android eseguiti.

In particolare con il comando:

```
$ emulator -system android_directory -kernel
   kernel_goldfish_image -show-kernel -shell
```

Qemu eseguirà la piattaforma indicata (kernel e stack Android) visualizzando nella shell le stringhe di debug del kernel e fornendo al termine del caricamento del sistema una interfaccia di comando con i privilegi dell'utente *root*.

2.4 Android e il Business

Da un rapporto ITU del settembre 2008[48] si apprende come il numero globale delle sottoscrizioni a contratti per dispositivi mobili sia arrivato a superare la cifra delle 4.000.000.000 unità. Tomi T Ahonen in un suo post[20] fornisce una descrizione molto interessante della situazione globale: a fronte di un numero così grande di dispositivi mobili attivi il numero di sottoscrizioni a contratti internet da locazioni fisse è dell'ordine dei 950 milioni (quattro volte di meno) mentre gli utenti attivi della rete globale si attestano nell'ordine delle 1.300 milioni di unità .

Il mercato dell'internet-mobile parrebbe attestarsi ad una base d'utenza di 350 milioni di unità , ma se si conta la sovrapposizione di utenze (una persona che possiede un computer connesso ad internet può possedere un telefono capace di navigare) questo numero può essere addirittura

maggiore. Ahonen stima a fine del 2008 che il numero di utenze mobili attive in internet per accessi browser sia pari a circa 1100 milioni di unità , superiore al numero di accessi via Personal Computer!

Google si inserisce in questo mercato con un sistema operativo capace di supportare gli standard di comunicazione in grado di portare i servizi di Google anche sui dispositivi mobili. L'obbiettivo che rende lo sviluppo di Android appetibile è quello di penetrare il mercato dei sistemi operativi per dispositivi mobili per affermare gli standard di comunicazione che permettono a Google di fare soldi attraverso la propria offerta di servizi.

In Android Google ha facilitato l'accesso ai propri servizi attraverso lo sviluppo di API native facendo questa una piattaforma privilegiata, ma l'obbiettivo principale è l'affermazione di una modalità di accesso alla rete standard per i normali PC che a fatica ora si sta cercando di riprodurre anche nei nuovi dispositivi mobili[24].

Capitolo 3

Microarchitettura ARM11

> In Informatica l'*Architettura Instruction Set Architecture (ISA)* definisce il set di istruzioni ed il modello di programmazione che ogni microprocessore deve implementare se basato su tale architettura.
> Le differenti implementazioni di una Architettura ISA possono variare in performance e funzionalità offerte ed essere ottimizzate per differenti applicazioni.
>
> Il modo in cui una Architettura ISA (Set di istruzioni e Modello di programmazione) viene implementata in un processore è detto *Microarchitettura*.
> Le implementazioni possono variare a seconda di obbiettivi specifici oppure adeguamenti ad innovazioni tecnologiche correnti.

La microarchitettura ARM11 è la prima implementazione del set di istruzioni ARMv6.

E' stata sviluppata seguendo le finalità e gli obbiettivi dell'architettura implementata: il target operativo è quello dei dispositivi embedded ed è stata posta attenzione particolare alla riduzione del consumo di potenza.

In questo capitolo, oltre ad una breve descrizione dell'evoluzione e caratteristiche innovative dell'architettura ISA ARMv6, verranno presentati i maggiori dettagli implementativi che caratterizzano la microarchitettura ARM11 ed infine verrà presentato il Core ARM1136JF-S: modulo di elaborazione posto all'interno dell'Application Processor presente nella board di sviluppo utilizzata.

Per maggiori informazioni si rimanda ai documenti: [25], [27], [45].

3.1 Architettura ARMv6

Evoluzione delle architetture ISA ARM

Il rapporto performance/potenza dissipata è stato uno dei fattori di maggior interesse nello sviluppo delle architetture ARM fin dalle prime versioni. Per questo nel tempo è stata data maggiore importanza alle tecniche di ottimizzazione: migliorando l'efficienza di utilizzo della memoria, il throughput di trasferimento dati con la stessa, ottenendo maggiori performance nelle operazioni numeriche ed accelerando compiti ricorrenti.

Di seguito sono presentate le maggiori innovazioni portate dalle precedenti architetture:

ARMv3: Indirizzamento a 32 bit e migliore gestione delle eccezioni.

- **Variante** T Set di istruzioni Thumb a 16 bit (utilizzo più efficiente della memoria e maggior throughput nel caricamento delle istruzioni).
- **Variante** M Registri risultato a 64 bit per moltiplicazioni tra grandi numeri (Standard nella versione v4).

ARMv4: Istruzioni di lettura e scrittura di halfword (16 bit).

ARMv5: Miglioramenti dello switch tra modalità normale e Thumb

- **Variante** E Set di istruzioni DSP per moltiplicazioni veloci ed operazioni aritmetiche con saturazione.
- **Variante** J Accelerazione per l'esecuzione nativa di bytecode Java.

Le varianti TEJ fanno parte dell'architettura ARMv6.

Gestione più efficiente della Memoria

Le performance di un sistema a microprocessore sono strettamente legate all'efficienza nell'utilizzo della memoria: parametri come il tempo medio di fetch di istruzioni e latenze di lettura/scrittura di dati incidono fortemente nella velocità di esecuzione risultante oltre al risparmio di potenza dissipata dovuto alla riduzione degli accessi.

Le innovazioni portate dalla versione 6 sono:

Cache TCM : oltre al normale sistema di Cache L1, è stata definita un'area di memoria strettamente accoppiata con il core interno (Tightly-Coupled Memory) gestita via software. Può essere utilizzata per caricare dati da elaborare in modo intensivo che non incorreranno nelle regole automatiche di aggiornamento della cache. Il trasferimento dati da e verso la Cache TCM avviene tramite due canali DMA attivi in modo esclusivo.

Translation Address Tag : Nel Tag che descrive la singola pagina di memoria presente in Cache L1 è stato aggiunto un campo di Indice del Processo software che ne detiene il possesso. Il risultato è la possibilità di un utilizzo concorrente della cache da parte di più processi eliminando la necessità di cache flush durante il context switch.

ARM Ltd conta in un miglioramento prestazionale fino al 30% in termini di throughput grazie a queste ottimizzazioni.

Raw Multiprocessing

Le funzionalità richieste dai sistemi portatili odierni spesso sfociano nella definizione di architetture multiprocessore dove ad ogni singolo core viene assegnata un compito specifico. Ad esempio per uno Smartphone un processore può gestire l'interfaccia utente mentre un secondo può agire come DSP per la gestione della comunicazione radio.

ARMv6 introduce in questo senso due nuove istruzioni per la lettura e scrittura esclusive di dati in memoria fornendo le basi per una gestione concorrente più efficiente della stessa e per metodi di sincronizzazione inter-cpu consistenti:

LDREX : esegue una lettura in memoria ed inizializza un monitor per "controllare" la locazione letta.

STREX : esegue una scrittura in memoria ritornando, nel registro risultato, un valore positivo se il monitor non ha osservato accessi concorrenti alla stessa locazione di memoria.

Raw Multimedia

Elaborare in modo più efficiente grandi quantità di dati permette di aumentare l'offerta di funzionalità multimediali: possono essere implemen-

tati codec audio e video avanzati con un miglior rapporto qualità /occupazione di banda e può essere migliorata la user experience inserendo grafica 3D nelle interfacce utente.

A questo proposito nell'architettura ARMv6 sono state introdotte istruzioni *Single Instruction Multiple Data (SIMD)* per aritmetica a 16 e 8 bit (addizioni, sottrazioni, moltiplicazioni con accumulatore ed altre). L'accelerazione ottenuta nelle applicazioni multimediali può arrivare fino a 2-4 volte le performance ottenibili con le architetture precedenti.

Bus dati più larghi

Sono state definite istruzioni a supporto di bus a 64 bit e maggiori, pur mantenendo l'architettura di elaborazione a 32 bit. Questa organizzazione vuole migliorare il throughput del sistema, aumentando la quantità di dati spostati in un singolo accesso, mantenendo l'efficienza energetica dell'architettura a 32 bit.

Eccezioni ed Interruzioni più veloci

Gli interrupt sono un meccanismo fondamentale per i sistemi odierni sia nelle implementazioni di tipo real-time, dove l'efficienza di gestione degli interrupt rappresenta un fattore critico dell'intero sistema sistema, sia nei normali sistemi a microprocessore dove miglioramenti nelle latenze degli interrupt comportano un aumento diretto delle performance globali.

Per migliorare la latenza agli interrupt sono state introdotte le seguenti modifiche:

Nuova modalità Fast Interrupt : settando il bit di stato FI nel registro CP15 viene abilitata la modalità Fast Interrupt dove anche le istruzioni di lettura e scrittura multiple (atomiche altrimenti) possono essere interrotte.

Gestione Vettoriale degli Interrupt : settando il bit VE nel registro CP15 viene abilitato il Controller Vectored Interrupt Controller (VIC) esterno per una gestione accelerata del processo di risposta all'evento.

Stack indipendenti : la nuova organizzazione dei registri permette di mantenere stack indipendenti per le diverse modalità operative eliminando l'overhead della gestione software per tali stack.

3.2 Innovazioni nella microarchitettura ARM11

Pipeline a 8 stadi

Il processo di elaborazione delle istruzioni è suddiviso ora in 8 stadi secondo lo schema in figura 3.1.

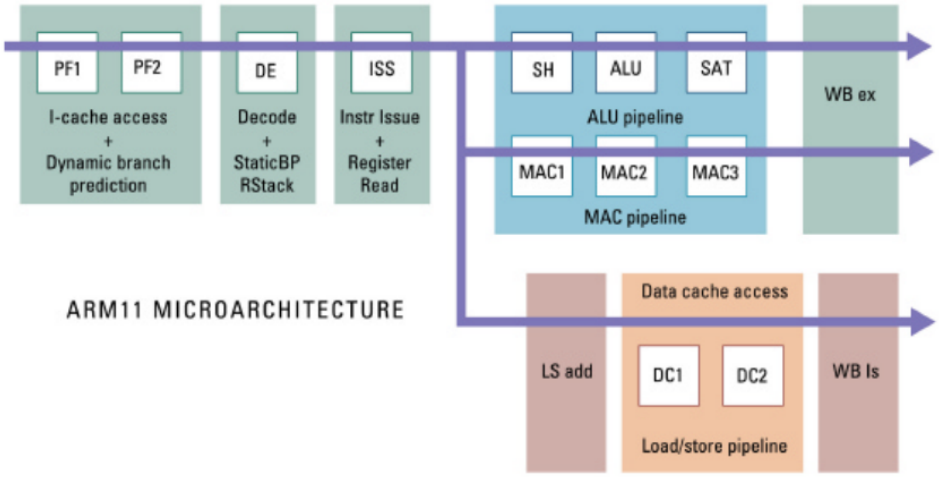


Figura 3.1: Stadi della Pipeline nell'architettura ARM11.

Nel tentativo di ridurre al minimo le inefficienze dovute all'"inceppamento" della pipeline si è agito su più fronti a seconda della causa.

Interdipendenza tra le istruzioni in esecuzione nella pipeline

Maggiori sono gli stadi della pipeline maggiori sono i cicli di clock di attesa affinché l'istruzione che determina il risultato necessario alla successiva, termini, permettendo quest'ultima di continuare il suo percorso di esecuzione. Per limitare i cicli di attesa è stato fatto uso esaustivo della tecnica del *forwarding*.

Istruzioni di salto condizionato

Nel caso di branch condizionali si può verificare lo stallo dell'intera pipeline dato che l'istruzione successiva da caricare dipende dal risultato della condizione del branch. Sono state implementate due tecniche di *branch prediction* statica e dinamica, che agiscono alternativamente a seconda della probabilità presunta di successo della previsione:

Dinamica - Viene utilizzato un semplice database di frequenza per le 64 istruzioni di branch più recenti dove, per ogni istruzione, vengono memorizzati quattro indirizzi target del branch separati nelle categorie *Strongly Taken*, *Weakly Taken*, *Strongly not Taken*, *Weakly not Taken*. L'istruzione successiva sarà caricata dall'indirizzo target del branch più frequentemente.

Statica - Se viene analizzata un'istruzione di branch non presente nel database la politica è: se l'istruzione salta all'indietro si assume la presenza di un loop riprendendo così l'esecuzione dall'inizio dello stesso (target del salto); altrimenti, si attende il risultato della condizione prima di continuare nel caricamento delle istruzioni.

E' implementata inoltre la tecnica di *branch folding* per la quale se il risultato della predizione è che il salto non viene eseguito, allora l'istruzione di branch viene eliminata dalla pipeline (folded) risparmiando così cicli macchina.

La somma di queste tecniche dà l'85% di branch previsti correttamente con un risparmio di circa cinque cicli macchina nel caso di branch folding.

Waiting time di accesso alla memoria

Leggere e scrivere nella cache in caso di cache-miss risulterebbe in uno stallo della pipeline pari al tempo necessario ad accedere alla memoria principale se il percorso di esecuzione è lineare.

Per ridurre questo ritardo l'architettura ARM11 prevede due percorsi di esecuzione (per istruzioni aritmetiche-logiche ed istruzioni di lettura/-scrittura in memoria) ed implementa tre tecniche per sfruttare il parallelismo creato: *non-blocking*, *hit-under-miss* ed *out-of-order completition*.

Se un'istruzione di lettura/scrittura dati provoca un cache-miss (il dato non è presente nella cache), grazie ai percorsi differenti seguiti (vedi la figura 3.1), questa viene "parcheggiata" in attesa che la lettura nella memoria principale sia terminata non fermando il flusso di esecuzione della pipeline (non-blocking) delle istruzioni non dipendenti da quella in attesa. Il flusso di queste istruzioni può continuare anche fino al completamento (out-of-order completition). Il flusso della pipeline non si blocca anche se l'istruzione successiva è ancora di lettura o scrittura (grazie ai due stadi nella Load Store Unit) permettendone la conclusione in caso il dato sia presente (hit-under-miss).

La struttura permette un massimo di due istruzioni di lettura/scrittura in attesa di completamento, una terza provocherà lo stallo della pipeline.

Micro TLB

La microarchitettura ARM11 implementa il modello di Cache L1 definito nelle specifiche ARMv6: cache indirizzata fisicamente (indirizzi di accesso fisici non virtuali) con una ricca quantità di attributi tra cui il tag di processo.

L'accesso tramite indirizzamento fisico richiede la presenza nel modulo MMU di una unità Translation Look-aside Buffer (TLB). Questa deve essere di dimensioni adeguate a quelle della cache costituendo così un'area considerevole on-chip da alimentare ad ogni accesso alla memoria veloce. Per migliorare l'efficienza energetica degli accessi in cache ARM11 estende questo meccanismo con una Micro-TLB: unità simile alla prima ma di dimensioni minori (contiene le 10 entry ad accesso più frequente). Allora il modulo MMU per accedere alla cache si servirà innanzitutto della Micro-TLB e poi, se non è stata trovata l'entry di traduzione da indirizzo virtuale a fisico, dell'unità TLB principale.

Bus a 64-bit

Per migliorare le performance del sistema mantenendo efficienza energetica la microarchitettura ARM11 inserisce bus a 64 bit tra le seguenti unità fondamentali:

ALU - I/D Cache - Fornendo la capacità di caricare due istruzioni alla volta oppure di scrivere o leggere due registri nella cache in un singolo ciclo di bus.

Coprocessore - ALU - Abilitando la possibilità di fornire due operandi da elaborare in un solo ciclo di bus.

3.3 Il Core ARM1136JF-S

Primo core della famiglia ARM11 sviluppato da ARM. Come la maggior parte dei prodotti ARM il core ARM1136JF-S viene licenziato come Intellectual Property (IP): ARM non produce fisicamente chip contenenti la

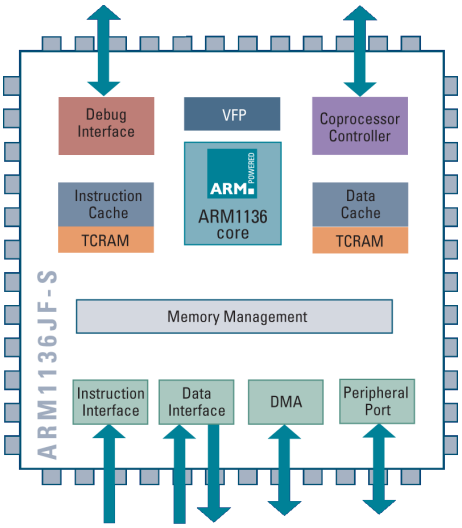


Figura 3.2: Diagramma a blocchi del core ARM1136JF-S

singola unità di elaborazione, ma fornisce il modulo in licenza per poterlo integrare nei processori prodotti dagli OEM.

La "**S**" nella dicitura enfatizza questo aspetto: indica la caratteristica *Synthesizable* cioè la capacità da parte degli OEM di personalizzare alcune proprietà del core come la dimensione delle cache oppure estenderlo collegando altri moduli attraverso il bus *AHB Lite*(versione single-master del bus ARM Advanced High-performance Bus) come la cache L2 (dati e istruzioni) ed altre periferiche compatibili.

Le lettere J ed F specificano invece funzionalità opzionali presenti in questo core:

J : indica la presenza dell'estensione Jazelle tramite la quale è possibile l'esecuzione diretta di bytecode Java. Per usufruire di tale estensione hardware devono essere presenti nel sistema operativo le librerie specifiche.

F : indica la presenza del Coprocessore vettoriale per calcoli in virgola mobile VFP.

Online è disponibile il manuale utente[1].

[1] `http://infocenter.arm.com/help/index.jsp`

Capitolo 4

L'Application Processor Freescale i.MX31L

Lo scopo di questo capitolo è quello di presentare l'Application Processor presente nella board utilizzata fornendo una breve descrizione dei moduli interni importanti per la trattazione successiva. Per maggiori informazioni si rimanda al contenuto del relativo manuale utente [33] e successivi aggiornamenti.

4.1 L'Application Processor

I processori i.MX31 e i.MX31L sono stati sviluppati da Freescale Semiconductor per dispositivi embedded portatili come smartphone, gaming machine etc. ambiti applicativi dove vengono richieste alte prestazioni multimediali a fronte di consumi di potenza ridotti.

Il design interno riprende la ricchezza di periferiche accessorie tipica delle architetture DSP adattata alle funzionalità richieste dai dispositivi portatili. Questa tipologia di architettura viene detta *Application Processor* dove si cerca di ottenere il miglior rapporto tra capacita di elaborazione + funzionalità offerte su potenza dissipata.

All'interno del chip troviamo allora il core di elaborazione ed un gran numero di periferiche addizionali in grado di soddisfare le necessità di connettività , multimedialità e capacità di archiviazione tipiche dei dispositivi embedded portatili, enumerate nella figura 4.1.

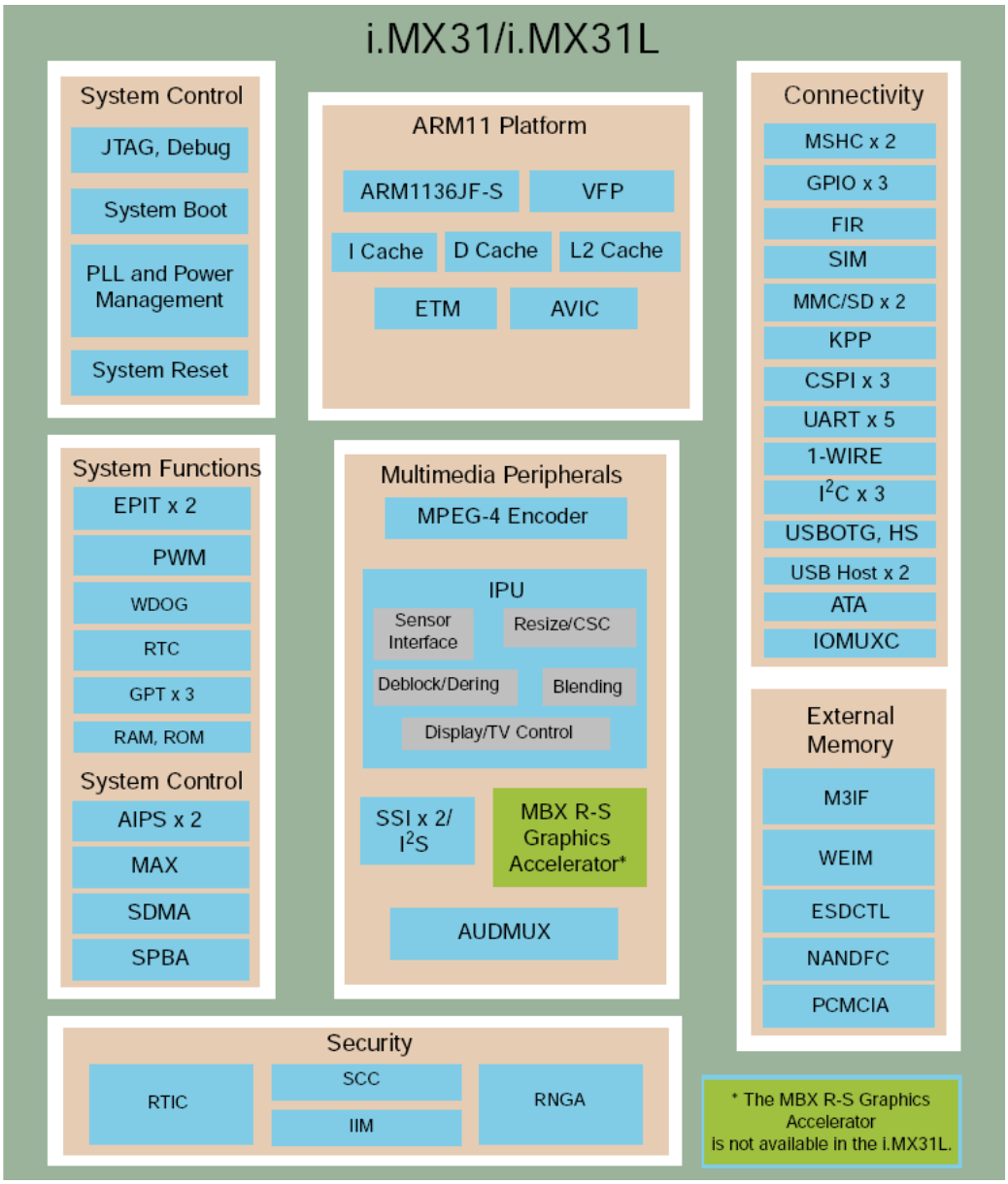


Figura 4.1: Diagramma a blocchi dell'Application Processor i.MX31

Il processore i.MX31L si differenzia dalla versione i.MX31 per l'assenza del dispositivo di accelerazione grafica 3D.

4.2 Il Core di Elaborazione

L'unità di elaborazione è data dal core ARM1136JF-S descritto nella sezione 3.3.

Realizzato con processo produttivo a 0.13 μm è stato dotato di:

- 16Kbyte di Data cache e 16 KByte di Instruction cache connesse attraverso bus a 64 bit.
- 128 KByte di cache L2 unificata (dati e istruzioni) connessa tramite il bus AHB Lite ed accessibile tramite tre interfacce:
  - read-only 64-bit instruction interface.
  - 64-bit bidirectional data read/write interface.
  - write only 64-bit data interface.
- 16 KByte di SRAM per applicazioni a basso consumo energetico.
- 32 KByte di ROM per il codice di bootstrap e dati.

La frequenza di funzionamento può variare nel campo 333-665MHz con prestazioni pubblicate fino a 660 Dhrystone[1], 2.1 MIPS.

4.3 La gestione degli interrupt

I processori i.MX31 ed i.MX31L dispongono di una periferica dedicata per la gestione degli interrupt detta *ARM11-platform Vectored Interrupt Controller (AVIC)* connessa al core ARM1136JF-S tramite la porta Vectored Interrupt Controller (VIC).

L'AVIC permette di gestire fino a 64 sorgenti di interrupt normali o veloci con priorità e permette una gestione accelerata in reazione ad interrupt normali: contiene un array di 64x30 word chiamato *Vector Table* nel quale è possibile definire, per ogni sorgente di interrupt abilitata, l'indirizzo della routine di gestione. In caso di interrupt l'AVIC segnalerà l'evento al core e trasmetterà sul bus il relativo indirizzo della routine di gestione.

Altre funzionalità importanti:

- Ogni sorgente può essere impostata come sorgente interrupt normale o veloce.
- Possiede un registro apposito per indicare eventuali interrupt in attesa.

[1] Benchmark computazionale che contiene sole operazioni su interi[30]

- Ogni sorgente di interrupt può essere abilitata o disabilitata indipendentemente (Utile per la gestione annidata degli interrupt).
- Fornisce un meccanismo per schedulare un interrupt via software.

4.4 Interfaccia verso la memoria esterna

L'interfaccia External Memory Interface (EMI) è in grado, data l'elevata adattabilità , di collegare una grande varietà di device di memoria: il bus dati può lavorare a 16 o 32 bit, è possibile abilitare la funzionalità di Address Interleaving per indirizzi di grandi dimensioni, sono presenti quattro porte di input con *Arbitration*, implementa un meccanismo di gestione di *Bus Master* alternativi (utile ad esempio se è presente un chip grafico collegato al bus di sistema) oltre che un meccanismo di gestione condivisa della memoria con device esterni.

Sono supportati device di memoria ad alta velocità come SRAM, SDRAM (fino a 133MHz), PSRAM (fino a 133MHz), DDR SDRAM(fino a 266MHz di data rate). Inoltre sono presenti i moduli di interfaccia verso dispositivi di memorizzazione di massa quali: dispositivi Flash NAND (supporto per chip a 8 e 16 bit fino a 2 GByte di spazio indirizzabile ed è presente un buffer interno di 2 KByte per l'accesso veloce), SmartMedia Card e PCMCIA release 2.1.

4.5 Clock e Power Management

I processori i.MX31 e i.MX31L possiedono un albero dei clock relativamente complesso capace di fornire ogni unità funzionale della frequenza di clock adatta (gestito tramite il modulo Clock Control Module (CCM)). Per migliorare l'efficienza energetica dell'Application Processor sono state implementate le due tecniche *"frequency scaling"* e *"power gating"*.

La frequenza di funzionamento della Microcontroller Unit (MCU) può essere adattata al carico di lavoro corrente (frequency scaling) grazie alla possibilità di eseguire lo switch run-time della sorgente alla radice del percorso di generazione ed il passaggio attraverso un clock-divider del segnale prima dell'ingresso nella MCU.

Per migliorare ulteriormente l'efficienza energetica sono stati definiti 3 domini di "Power gating" indipendenti: Piattaforma ARM11 (Core ARM11

+ MMU + Caches), DPLL(Clock management) e Periferiche. Per ogni dominio esistono le modalità OFF (dominio spento), Active (funzionamento normale) e Standby (L'energia viene mantenuta al minimo possibile).

4.6 Multiplexing, GPIO, e Pad Control

I segnali di interfaccia dei device interni all'Application Processor sono mappati sui pin di collegamento fisici attraverso un meccanismo di multiplexing. Così, a seconda dell'implementazione, è possibile abilitare o meno le porte di collegamento di determinati dispositivi e scegliere, tra le configurazioni possibili, la mappatura segnali/pin migliore[33, cap 4].

I circuiti logici che gestiscono il multiplexing permettono inoltre di impostare il comportamento elettrico dei singoli pin (pull-up, pull-down, isteresi etc..) detto Pad Settings e di eseguire le osservazioni necessarie per la generazione di interrupt da parte dell'AVIC. E' possibile inoltre impostare determinati pin come *General Purpose Input/Output (GPIO)* per modificare o acquisire via software lo stato dei singoli oppure per creare sorgenti interrupt.

4.7 Interfaccia AIPS verso il bus di espansione

L'interfaccia *AHB-Lite 2 to IP Bus Rev 3.0 SkyBlue line interface (AIPS)* fornisce la possibilità di collegare al bus di sistema AHB-Lite dispositivi a banda limitata conformi allo standard Freescale SkyBlue line.

Sono presenti due moduli di interfaccia AIPS: AIPS-A ed AIPS-B a 32 bit con 32 MByte di spazio di indirizzamento ciascuno. Sono supportate letture e scritture di byte, half-word, word e double-word con buffer permodulo dedicati e che richiedano almeno due cicli di clock per la lettura e tre cicli di clock per la scrittura.

4.8 Shared Peripheral Bus Arbiter (SPBA)

L' SPBA è un modulo di interfaccia tre a uno per bus SkyBlue line: capace di arbitrare l'accesso al singolo bus delle periferiche da parte di massimo tre linee di controllo con un meccanismo di resource locking della singola periferica al master bus corrente.

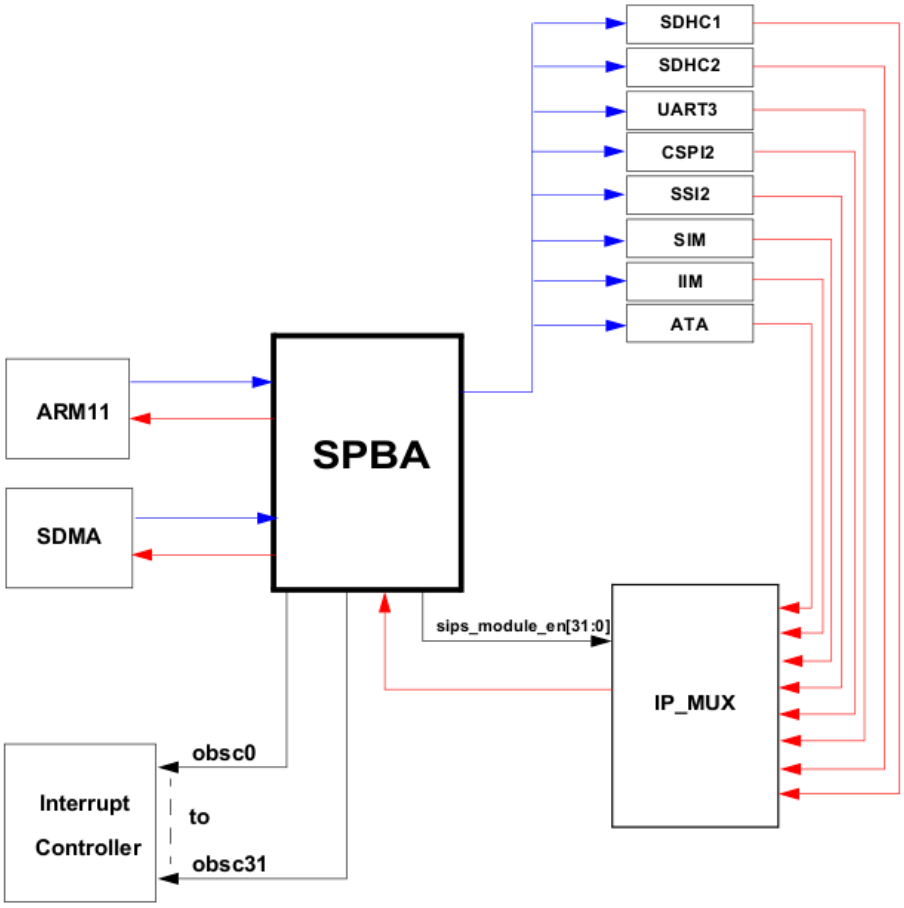


Figura 4.2: Shared Peripheral Bus Arbiter (SPBA).

Dato che il bus SkyBlue line è di tipo single master l'accesso contemporaneo da parte della Microcontroller Unit ed il modulo SDMA alle periferiche deve avvenire tramite un'interfaccia capace di multiplarne adeguatamente le attività . Il modulo SPBA da la possibilità di accedere ai master bus fino a 31 periferiche in modo condiviso (il modulo SPBA può essere considerato la 32'esima periferica) con una frequenza di funzionamento fino a 67 MHz.

4.9 Smart Direct Memory Access Controller

Modulo di gestione del sottosistema DMA capace di liberare il core centrale dalle operazioni di trasferimento dati sequenziali tra memoria e memoria o tra periferiche on-chip e memoria.

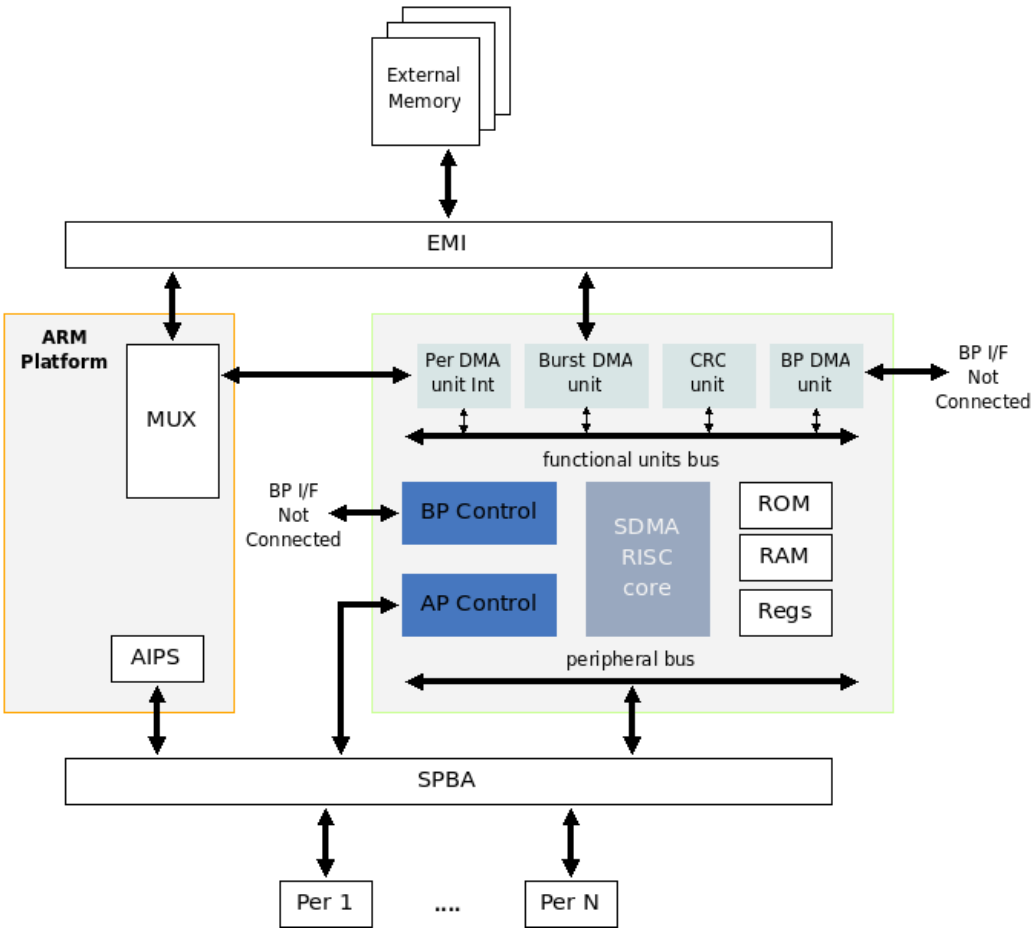


Figura 4.3: Connessioni del modulo SDMA.

Il modulo SDMA è costituito da un processore RISC completo connesso al core ARM ed al bus delle periferiche come in figura 4.3. Sono disponibili 32 canali DMA virtuali con scheduling preemptive priority based a due livelli di priorità . La dimensione del singolo burst di trasferimento è programmabile fino a 16 word ed è presente un meccanismo di timeout con error firing attraverso interrupt se non è possibile concludere il burst nel tempo definito.

4.10 Timers GPT ed EPIT

Gli Application Processors i.MX31 e i.MX31L includono due tipologie di timer:

General Purpouse Timer (GPT)

Contatore a 32bit con clock source selection (tra cui una sorgente esterna) programmabile per essere attivo anche in low-power mode. Sono presenti tre registri compare associati ognuno ad un pin di output programmabile (l'evento di match può: invertire lo stato del pin, portarlo al livello alto o basso, oppure generare un impulso) e due segnali di input capture con trigger capaci di far memorizzare lo stato del contatore in due registri appositi.

Le modalità di funzionamento supportate sono:

Restart mode : possibile per il solo registro di compare 1, il contatore ricomincia da 0x0 dopo l'avvenuto evento Compare1. Scrivere nel primo registro compare in questa modalità produce il reset del contatore.

Free-run mode : il contatore continua il conteggio in modo indefinito fino all'evento di rollover da dove riprende da zero (0xFFFFFFFF $->$ 0x00000000).

Il GPT può generare interrupt negli eventi capture, compare e rollover (reset per overflow).

Enhanced Periodic Interrupt Timers (EPIT)

Due timer EPIT "set and forget" a 32 bit capaci di generare interrupt ad intervalli regolari con il minimo intervento software. Possono essere programmati per essere attivi in modalità low-power.

Entrambi si basano su un contatore inverso e possono funzionare in due modalità :

Set and forget : il contatore, raggiunto lo zero, ricarica automaticamente il valore programmato in precedenza nel registro EPITLR senza la necessità di intervento specifico via software.

Free-run mode : il contatore continua il conteggio in modo indefinito fino allo zero per poi riprendere da 0xFFFFFFFF. Anche in questa modalità è possibile programmare il reset ad un valore specifico tramite il registro EPITLR.

L'interrupt, se abilitato, viene generato ogniqualvolta il timer raggiunge lo zero.

4.11 Interfacce seriali UART

Sono presenti 5 moduli Universal Asynchronous Receiver-Transmitter (UART) programmabili, capaci di trasmettere dati nelle configurazioni seguenti:

- Data word lunghe 7-bit o 8-bit , 1 o 2 bit di stop, parità definita tra: (even, odd oppure nessuna).
- Baud rate programmabile fino ad un massimo di 1.875 Mbit/s
- 32-byte di buffer di trasmissione FIFO ed in ricezione buffer FIFO di 32 halfword con supporto auto-baud.
- Supporto alla trasmissione infrarossi Infrared Data Association (IrDA) 1.0 nelle modalità Serial Infrared Speed (SIR).

4.12 Interfaccia I^2C

E' presente un controller per bus multiple master *Philips* Inter Integrated Circuit (I^2C).

Il bus a due fili (uno di clock ed uno per dati) bidirezionale (la linea dati è di I/O) I^2C fornisce un metodo semplice per lo scambio di dati tra componenti della stessa board. Sullo stesso bus possono essere connessi più componenti ai quali viene assegnato un indirizzo differente che possono agire tutti alternativamente come bus-master. Il controller presente supporta i protocolli di arbitration e collision detection delo standard I^2C.

La velocità di clock può essere selezionata tra le 64 possibili fino ad arrivare 400Kbit al secondo.

4.13 Interfaccia USB

Il modulo USB comprende tre porte di collegamento: USBH1,USBH2 e USBOTG, delle quali USBH1 e USBOTG possono essere attive in modo esclusivo dato che condividono una parte importante dei segnali di interfaccia del modulo. Tutte le porte sono USB 2.0 complaint: implementando lo standard Intel Enhanced Host Controller Interface (EHCI)[41] per le due porte USBH1 e USBH2 ed il supplemento allo standard USB 2.0 On The Go (OTG) per la porta USBOTG. Le velocità di trasferimento

supportate sono quelle definite dallo standard USB 2.0: high speed (480 Mbit/s), full speed (12 Mbit/s) e low speed (1.5 Mbit/s).

Per raggiungere la compatibilità con lo standard USB 1.1 è presente un "Embedded Transaction Translator" modulo hardware interno ai controller trasparente all'interfaccia EHCI che permette la connessione diretta di dispositivi USB 1.1 senza la necessità di moduli "Companion Controller".

Il design del modulo è tale da supportare connessioni transceiver-less verso dispositivi on-board che implementano l'interfaccia ULPI. La connessione del modulo a porte USB fisiche necessita la presenza di dispositivi transceiver che a partire dall'interfaccia ULPI implementino il livello fisico USB (PHY level).

USB in pillole

Alberi USB. In riferimento allo standard USB 2.0 [32] il Bus USB viene utilizzato per connettere un *Host* (PC, workstation ..) ad un certo numero di *Device* periferici. I due ruoli sono ben definiti e non modificabili successivamente l'instaurazione della connessione.

Nell'albero delle connessioni USB è presente un'unica radice (*Host* come system master), alla quale può essere connesso una foglia (dispositivo *Device*) oppure un nodo intermedio (*Hub*) capace di connettere foglie o altri nodi intermedi.

I PC moderni supportano diversi alberi USB, solitamente un albero USB 2.0 (alla velocità di 480 Mbit/sec) ed alcuni alberi USB 1.1 (12 Mbit/sec ciascuno) quest'ultimi creati alla connessione di almeno un device USB 1.1.

Host Controllers. Viene così chiamata la porzione hardware che implementa le funzionalità USB Host. Sono stati sviluppati tre standard di interfaccia verso Host controllers nell'evoluzione di standard USB:

UHCI e OHCI : Universal Host Controller Interface e Open Host Controller Interface, standard sviluppati per USB 1.1. Supportano le due velocità "Low Speed" 1.5 Mbit/sec (192 KByte/sec) e "Full Speed" 12 Mbit/sec (1.5 MByte/sec).

EHCI : Enhanced Host Controller Interface, standard sviluppato da Intel per controller USB 2.0. Supporta la velocità di trasferimento "High

Speed" 480 Mbit/sec (60 MByte/sec) propria dello standard USB 2.0.

Controller USB 2.0 che implementano lo standard EHCI raggiungono la retrocompatibilità con i dispositivi USB 1.1 solamente se:

- Dispongono di almeno un "*Companion Controller*" UHCI o OHCI, moduli hardware interni al Controller USB con registri di interfaccia indipendenti dal Controller EHCI che prendono il controllo del bus USB nel caso venga connesso un device USB 1.1.
- Viene connesso un Hub USB 2.0 che implementa al suo interno il modulo "*Transaction translator*" capace di incapsulare le due modalità di connessione USB1.1 nella modalità "High Speed".

Device Controllers. Viene così chiamata la porzione hardware che implementa la funzionalità USB Device secondo le modalità di trasferimento e le classi di dispositivo definite dagli standard USB 1.1 e USB 2.0.

USB OTG. USB On The Go, addendum allo standard USB 2.0 è stato sviluppato specificatamente per device embedded come PDA etc.. definisce le specifiche hardware e di interfaccia per controller capaci di agire sia come *device* che come *host* nella comunicazione USB. Successivamente alla definizione del ruolo per una data connessione, è chiaro che questo rimarrà tale per l'intera durata della connessione stessa.

A seconda delle necessità un device embedded può assumere i ruoli di device oppure host: se un PDA viene connesso al PC per la sincronizzazione allora necessariamente il PC avrà la parte di Host ed il PDA dovrà comportarsi come un normale device, mentre se si vuol connettere una memoria USB al PDA allora questo dovrà assumere il ruolo di Host.

4.14 Secured Digital Host Controller (SDHC)

Il modulo SDHC fornisce il supporto per la connessione di dispositivi Multi Media Card (MMC) e memorie Secure Digital (SD) includendo SD I/O combo card (dispositivi con funzionalità di storage ed I/O).

4.15 Interfaccia PCMCIA

La porta PCMCIA Rel.2.1 presente nell'Application Processor i.MX31L è parte del modulo External Memory Interface (EMI) e fornisce supporto alla connessione di periferiche esterne quali schede di rete wired/wireless, Compact Flash Card ed altre.

4.16 Il modulo Watchdog

Il modulo Watchdog fornisce un metodo per recuperare il controllo del sistema nel caso questo non risponda per un determinato periodo di tempo (Periodo di Watch). Entrambe le variabili periodo di controllo e azione eseguita sono programmabili:

Periodo di Watch : Programmabile nel campo $[0.5, 64]$s con risoluzione 0.5s.

Azione al termine : In caso di timeout il modulo watchdog può essere programmato per eseguire le seguenti azioni in modo incrementale:

1. Generare un interrupt per gestire l'evento via software.
2. Reset dell'Application Processor dopo un periodo di timeout successivo all'istante di generazione dell'interrupt.
3. Reset dei device on-board attraverso un pin esterno in seguito al reset dell'Application Processor.

4.17 Real Time Clock

Il modulo Real Time Clock (RTC) serve a mantener aggiornato l'orologio di sistema anche nello stato power-down. Oltre alla funzionalità di orologio ha la capacità di:

- Generare un interrupt dopo un numero programmato di minuti.
- Generare un interrupt (allarme giornaliero) ad un'ora programmata.
- Generare interrupt: uno-al-giorno, uno-all'ora, uno-al-minuto, uno-al-secondo

4.18 Immagini, Video e Grafica

Gli Application Processor i.MX31 e i.MX31L presentano una catena di gestione dei contenuti multimediali completa: sono capaci di acquisire, manipolare e visualizzare flussi video con accelerazioni hardware dedicate. La figura 4.4 descrive l'intera catena di video processing.

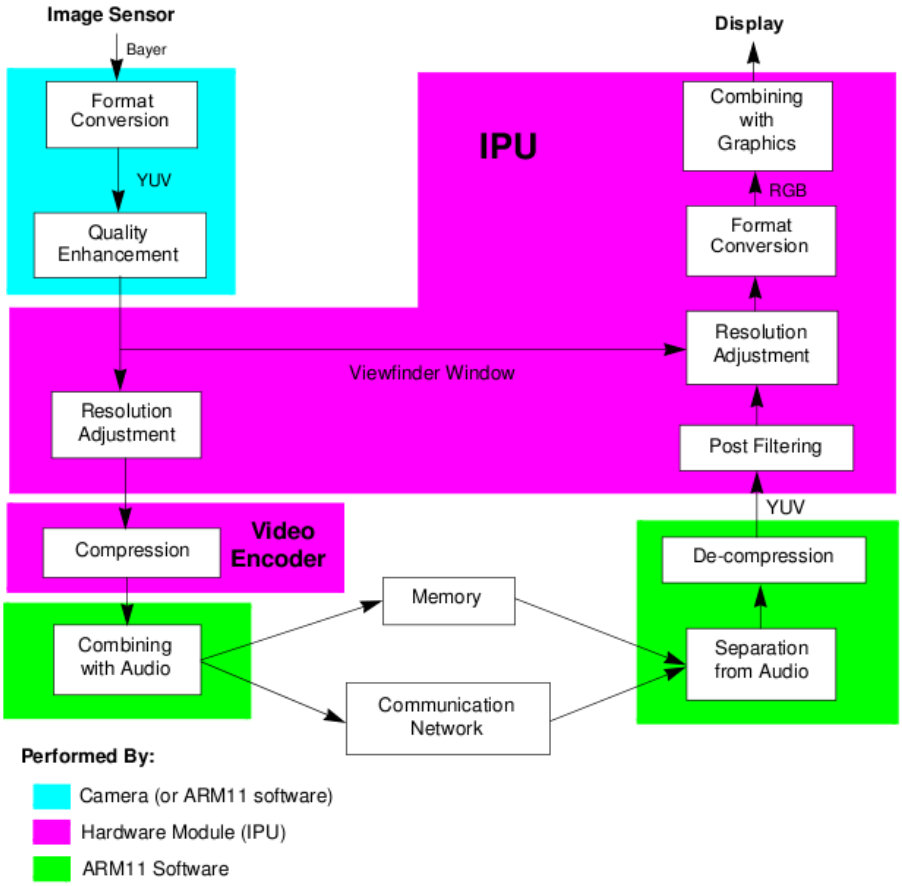


Figura 4.4: IPU workflow.

Il modulo Image Processing Unit (IPU) ha un ruolo centrale in questa catena: include le interfacce di acquisizione e visualizzazione dei contenuti video, oltre a moduli di hardware video-processing per liberare la CPU dal carico di operazioni standard (e non come il pre-processing per la compressione video MPEG4).

Di seguito sono presentate le unità contenute:

Camera Sensor Interface (CSI) : interfaccia di acquisizione da sorgen-

ti video. Si avvale del protocollo I^2C per collegare il chip della videocamera.

Image Converter (IC) : unità capace di effettuare diverse operazioni al frame ricevuto dalla CSI, in particolare:

- Resize del frame incrementando o diminuendone la dimensione (interpolazione lineare o decimazione pesata).
- Flip orizzontale del frame.
- Due livelli di conversione dello spazio dei colori RGB to YUV o viceversa.
- Tra i due livelli di conversione dello spazio dei colori è possibile combinare linearmente il frame con altre immagini.
- Rotazione del frame in multipli di 90°.

Post-Filter (PF) : elabora il frame in preparazione alla compressione MPEG4 effettuata dal modulo dedicato.

Synchronous Display Controller (SDC) : controller per display sincroni quali TFT, Smart display,Sharp display, TV-encoders (necessitano dei segnali di sincronismo VSYNC e HSYNC generati dall'unità DI). Contiene un blocco di image combining tra due livelli grafici (Background e Foreground).

Asynchronous Display Controller (ADC) : controller per display asincroni (display che dispongono della logica e memoria necessaria per visualizzare autonomamente la grafica) capace di pilotare fino a 3 display simultaneamente.

Display Interface (DI) : interfaccia capace di collegare fino a 4 display contemporaneamente in time-multiplexing. Converte dati provenienti dai controller SDC, ADC e dalla MCU (accesso raw) nel formato specifico dei display connessi secondo le specifiche programmate.

Image DMA Controller (IDMAC) : tutte le operazioni che hanno a che fare con buffer di memoria e tra unità interne nel modulo IPU vengono gestite in modalità DMA tramite la programmazione di questa unità .

Sono gestiti i canali DMA dai buffer video ai controller SDC e ADC, i canali di comunicazione con l'unità IC per i dati necessari ai vari livelli di processing ed infine i canali di comunicazione con l'unità PF per interventi nelle operazioni di post-processing per la compressione video MPEG.

4.19 Interfaccia Audio

Il sottosistema audio degli Application Processor i.MX31 e i.MX31L è costituito da due moduli di interfaccia per la connessione a processori audio esterni (gli standard di collegamento sono Synchronous Serial Interface (SSI) o Inter-IC Sound (I^2S)) connessi al bus interno attraverso un modulo di routing dinamico AUDMUX. E' supportato lo standard AC-97

Capitolo 5

Atmark Armadillo 500

Prodotta dalla ditta Giapponese Atmark Techno[1] la Board Armadillo 500 può essere considerata una buona piattaforma di sviluppo per sistemi embedded multimediali su architettura ARM11.

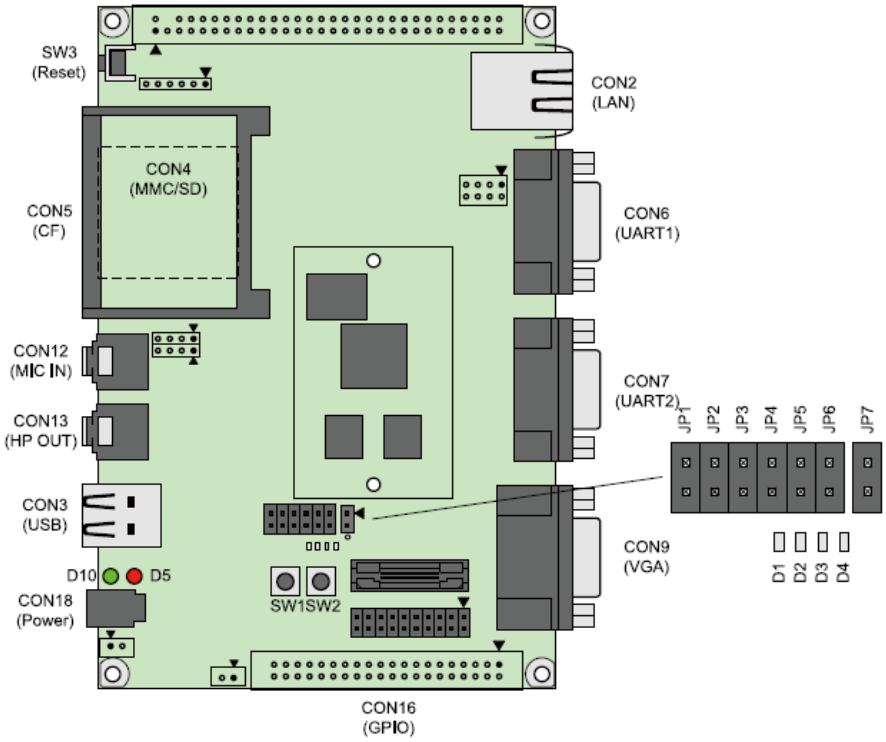


Figura 5.1: La board Atmark Armadillo 500

[1] http://www.atmark-techno.com

In questo capitolo verrà presentato un estratto utile della documentazione disponibile nel sito Atmark[23], tradotta dal giapponese grazie al servizio *Google Translate*[2].

Oltre a ciò che è qui presentato, nel proseguo del documento si farà riferimento al materiale contenuto nella directory `document/hardware/` del cd allegato alla board, contenente gli schemi logici della mother-board ed i datasheet di ogni componente utilizzato[22, 3].

In appendice A è riportata la mappatura nello spazio di indirizzamento dell'Application Processor i.MX31L delle interfacce di controllo per ogni dispositivo presente nella piattaforma.

5.1 L'hardware della board nel dettaglio

La figura 5.2 presenta il diagramma a blocchi della piattaforma, dove possono essere individuati i componenti principali descritti in seguito in maggior dettaglio.

5.1.1 Armadillo 500 SoM

Il modulo CPU contiene l'Application Processor Freescale i.MX31L descritto nel capitolo 4, un modulo SDRAM DDR da 64 MByte e 16 MByte (128 Mbit) di memoria NOR flash della famiglia Intel P30 (128P30B).

La connessione alla scheda madre avviene tramite due connettori FX10A-140S/14-SV (Hirose Electric) a 156 pin ciascuno.

5.1.2 Porte seriali RS232

Sono presenti due porte seriali RS232 standard connesse ai primi due moduli UART dell'Application Processor i.MX31L attraverso interfacce "channel driver" per raggiungere la compatibilità elettrica dei segnali con lo standard di canale RS232 [22, foglio 6].

5.1.3 Rete Ethernet

Connesso al bus di espansione è presente un controller SMSC LAN9118 per reti Ethernet 10BASE-T/100BASE-TX[22, foglio 3].

[2] `http://translate.google.it`

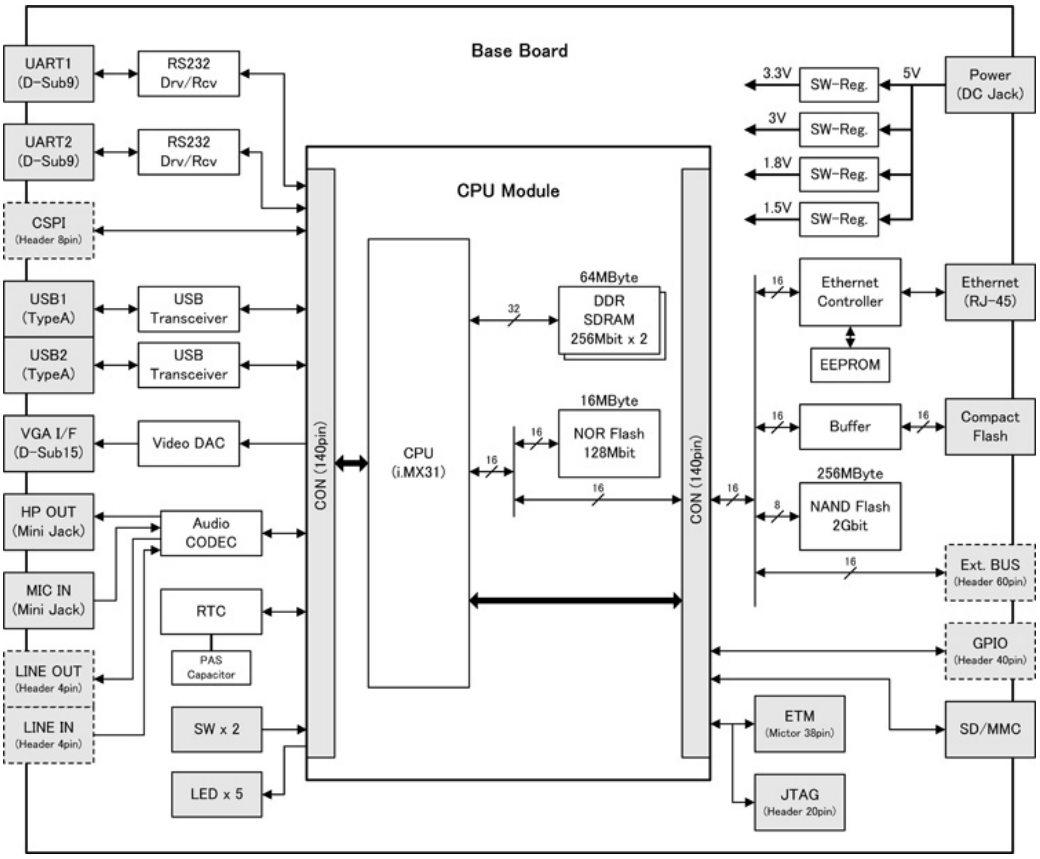


Figura 5.2: Diagramma a blocchi della board Atmark Armadillo 500.

5.1.4 Slot SD/MMC

E' presente uno slot SD/MMC per SD-Card connesso direttamente ai segnali del primo modulo SDHC dell'Application Processor i.MX31L (vedi la sezione 4.14).

I segnali dell'interfaccia SD *Card Detect* e *Write Protect* sono connessi all'Application Processor tramite pin GPIO dedicati [22, foglio 4].

5.1.5 Output Video

E' presente un connettore D-Sub15 pin per cavi RGB standard verso display.

I segnali prodotti dall'interfaccia digitale SDC dell'Application Processor sono convertiti nello standard analogico attraverso il chip Video DAC

(Digital to Analog Converter) Analog Devices ADV7125 [22, foglio 7].

5.1.6 NAND Flash

Sul retro della board è montato un chip ST Micro NAND Flash da 256 MByte con bus a 8 bit, organizzato in pagine da 2KByte connesso al modulo NAND controller dell'Application Processor i.MX31L [22, foglio 2].

5.1.7 Real Time Clock (RTC)

L'orologio di sistema è mantenuto da un circuito on board basato sul chip Seiko Instruments S-35390A collegato all'AP tramite bus I^2C ed alimentato da una batteria di backup per mantenerne la funzionalità anche dopo la disconnessione dell'alimentazione [22, foglio 6].

5.1.8 Tasti e LED

Sono presenti due tasti *"low active"* di input e 4 led di output connessi tutti a pin GPIO [22, foglio 6].

5.1.9 USB

Sono presenti due connettori USB di tipo Host connessi attraverso due differenti moduli transceiver NXP ISP1504 ai controller USBOTG e USBH2 dell'Application Processor [22, foglio 4].

5.1.10 Audio

Sono presenti i connettori Headphone Out e Mic In per l'I/O Audio connessi alla porta audio in modalità I^2S attraverso il modulo codec della Texas Instruments TLV320AIC23 con amplificatore interno [22, foglio 8].

5.2 Il boot loader

Hermit At (Versione di U-Boot personalizzata da Atmark) è il boot loader utilizzato nelle board Atmark che risiede all'inizio della memoria NOR flash del modulo CPU .

Capace di eseguire le inizializzazioni necessarie della board armadillo 500 è compatibile con le specifiche di boot del sistema operativo Linux.

5.2.1 Canale di comunicazione e controllo

Per interagire nella sequenza di boot è necessario collegare la workstation di sviluppo alla prima porta seriale della board ed utilizzare un software adatto come *minicom*.

Le impostazioni della connessione seriale sono le seguenti:

Parametro	**Impostazione**
Velocità di trasferimento :	115,200 bps
Lunghezza dei dati :	8bit
Bit di stop :	1bit
Parità :	No
Controllo di flusso hardware:	No

5.2.2 Contatti di configurazione

Detti anche JANPAPIN sono una serie di Jumper posizionati sotto il modulo del processore JP1-JP7.

I primi 6 Jumper sono utilizzati come input nel processo di boot:

JP1 indica al boot loader Hermit AT se presentare il menu di configurazione.

JP1	**Modalità**
Open	Dopo l'accensione il kernel viene eseguito automaticamente.
Short	Dopo l'accensione viene presentato il prompt dei comandi Hermit attraverso la connessione seriale.

Nel caso il boot loader sia stato compromesso, la configurazione seguente

dei jumper JP3-JP6 permette di abilitare la modalità UART boot del core ARM1136JF-S:

JP3	**JP4**	**JP5**	**JP6**	**Mode**
Open	Open	Open	Open	Boot normale
Short	Short	Open	Short	UART boot

5.2.3 Comandi fondamentali

Cortocircuitando il Jumper JP1, durante il processo di boot, il boot loader Hermit AT presenta attraverso la porta seriale l'interfaccia di controllo Hermit.

Qui di seguito i comandi fondamentali della CLI Hermit AT:

tftpdl

Indica ad Hermit AT di instaurare una connessione Trivial ftp attraverso la rete per scaricare un determinato file immagine dalla workstation e scriverlo nella regione di memoria indicata. Per poter eseguire questo comando la board deve essere connessa alla rete di sviluppo attraverso l'interfaccia ethernet:

```
hermit> tftpdl 192.168.0.10 192.168.0.2 --userland=linux.bin.gz
```

I parametri del comando hanno il seguente significato:

1. Indirizzo IP utilizzato dalla board.
2. Indirizzo IP del server TFTP (workstation).
3. Path del file da scaricare nella regione indicata della memoria NOR Flash (`-regione=path`). Il path è relativo rispetto alla directory root del server TFTP.

Il nome di regione identifica una delle seguenti partizioni della memoria NOR Flash del modulo CPU:

Regione	Descrizione	Dimensione
all	Per sovrascrivere l'intera memoria.	16 MB
boot loader	Prima area, contiene Hermit At. Errori di scrittura in quest'area richiedono una procedura di recovery particolare.	128 KB
kernel	Regione dove risiede il kernel in formato zImage.	2 MB
userland	Può contenere un file ramfs in formato compresso.	13.75 MB
config	Area ideata allo scopo di memorizzare i file di configurazione del sistema tra boot differenti.	128 KB

Tabella 5.1: Partizioni della memoria flash NOR del modulo CPU.

Di seguito è presentato il log di una normale trasmissione tftp[3]:

Listing 5.1: Log di una normale trasmissione tftp.

```
hermit> tftpdl 192.168.0.10 192.168.0.2 —kernel=linux.bin.gz

Client: 192.168.0.10
Server: 192.168.0.2
Region(kernel): linux.bin.gz
initializing net-device...OK
Filename : linux.bin.gz

................................................................
................................................................
..............................

Filesize : 1841551
programing: kernel

###############

completed!!
```

setenv e clearenv

Con il comando **setenv** è possibile impostare la stringa dei parametri del kernel in questo modo:

[3]Alle volte la connessione tftp si blocca prima della fase di download, solitamente eseguire un ping all'indirizzo della macchina target sblocca la situazione.

```
hermit> setenv console=ttymxc0 root=/dev/mmcblk0p1 rootdelay=1
   loglevel=7
```

Mentre il comando **clearenv** azzera la stringa dei parametri memorizzata.

boot

Dà l'ordine al boot loader di procedere con la sequenza di boot del kernel.

5.3 Compatibilità tra revisioni successive del SoM Armadillo 500

Il Jumper JP7 è utilizzato per indicare alla main-board quale è la versione del modulo CPU montata:

Versione Armadillo SoM	Stato JP7
A50**-U** A50**-U**B(A50**ZB)	Short
A50**-U**C(A50**ZC)	Open

In questo modo viene mantenuta la compatibilità elettrica tra mainboard e differenti revisioni del modulo CPU.

Capitolo 6

Linux su piattaforma Freescale i.MX31L

Linux è uno dei maggiori progetti open-source esistenti (per antonomasia *IL* progetto open-source), impegna una comunità vastissima di sviluppatori con diverse capacità e professionalità, regolata in gruppi specializzati nel mantenimento ed evoluzione di ogni aspetto del kernel.

Sin dalla prima versione pubblicata nel 1991, Linus Torvalds è ancora oggi per Linux un leader attivo in modo importante sia per la definizione degli obbiettivi sia nella gestione e sviluppo dei contributi al progetto.

Il sito internet *kernel.org*[1] è la vetrina che presenta il codice ufficiale del kernel Linux detto anche *mainline* o versione *vanilla*, mantenuto dall'intera comunità Linux. Ogni contributo al kernel da parte di un singolo sviluppatore, passata la procedura di review, viene inserito in questo repository e diventa parte del kernel ufficiale Linux.

Il processo di review assicura la qualità del mainline kernel, effettuato all'interno dei gruppi della comunità specializzati negli aspetti toccati dal contributo ed approvato dai leader degli stessi gruppi (persone che hanno guadagnato importanza grazie al proprio operato e che gestiscono attivamente i repository locali per gruppo del kernel Linux), evidenzia il più possibile problemi presenti nel singolo contributo dalle semplici sviste nel *Coding Style*[10] ad errori nelle funzionalità, nella sicurezza e possibili regressioni, fino alla discussione di nuove proposte (RFC) per intervenire in modo più profondo nelle funzionalità del kernel.

[1] `http://www.kernel.org/`

E' per questo che solo il codice mainline deve essere considerato di qualità a differenza di branch distribuiti da altre fonti, modificati per accogliere capacità specifiche, difficilmente mantenibili a fronte dello sviluppo principale del mainline kernel e quindi spesso destinati alla fossilizzazione in una specifica e datata versione del kernel.

Ciò che spinge la nascita di repository non ufficiali nel caso di aziende nel campo embedded è il miraggio della velocità di sviluppo di un sistema utilizzabile adatto al nuovo prodotto in cantiere. Il processo di review delle patch al main line kernel è si efficace in termini di qualità, ma richiede un tempo ritenuto troppo grande rispetto alle necessità di deployment dettate dal mercato ed impone un elevato rigore rispetto a personalizzazioni esotiche dovute ad hardware non conforme agli standard accettati.

Alcune aziende allora preferiscono sviluppare internamente tutto il codice di supporto in un repository locale creando una nuova linea di biforcazione (branch) la quale solitamente raggiunge una distanza tale dal kernel mainline che la fusione tra le due al termine dei lavori è ritenuta troppo costosa (oppure impossibile se suporta hardware non standard).

Oltre a non contribuire attivamente al progetto principale (privandolo di contributi che potrebbero essere utili anche ad altri membri della comunità), queste aziende si ritrovano nella condizione di non implementare quei bug-fix o nuove funzionalità date dalle versioni successive del mainline kernel nel proprio branch se non con gli stessi costosi processi di merge inverso risultando così in un prodotto potenzialmente incompleto e poco mantenibile[28].

Questo è il caso del branch Atmark del kernel Linux a supporto della board di sviluppo Armadillo 500[2]. Atmark mette a disposizione due versioni del kernel (2.6.18 e 2.6.26 quest'ultima pubblicata solo di recente, non presente all'inizio di questo lavoro di tesi) basate sul branch Freescale a supporto del core i.MX31[3] con modifiche ed estensioni nei driver di periferica.

Tenendo conto che la BSP Freescale non è mai stata inserita nel repository mainline perché non ritenuta di qualità sufficiente e che nella versione 2.6.27 è stato iniziato un restyling completo del codice a supporto delle board ARM, il codice sviluppato da Atmark ad oggi è diventato obsoleto

[2]Atmark directory `http://download.atmark-techno.com/armadillo-500/`

[3]Linux BSP for Freescale i.MX31ADS all'indirizzo: `http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=0127260061033202A7`

ed il supporto ai nuovi kernel per questa board è decaduto.

Uno degli obbiettivi di questo lavoro di tesi è sviluppare il codice di supporto alla board Atmark Armadillo 500 in seno alla comunità che mantiene la platform per i processori Freescale i.MX (famiglia MXC), via necessaria per l'inserimento di questo contributo nel mainline kernel.

6.1 Il progetto ARM-Linux

Il primo porting di Linux su processore ARM è stato eseguito con successo da Russel King nel 1994. Da allora fino ad oggi Russel ha continuato il suo contributo nelle vesti di *mantainer* ufficiale per questo progetto e punto di riferimento per i differenti sotto gruppi di lavoro.

Il progetto ARM-Linux è ora di dimensioni considerevoli, sono supportate le ARM ISA dalla v3 alla v7, 29 core specifici, 39 architetture di Application Processor (AP) e 229 board.

I contenuti del sito ufficiale[4] non sono del tutto aggiornati specie nella documentazione ferma al 2004 ma costituisce il punto di accesso alle mailing list, vero strumento di interazione con i membri della comunità arm-linux per lo scambio di informazioni ed il processo di review delle modifiche proposte ai sorgenti del progetto.

Esistono altri siti web più o meno aggiornati che documentano il progetto ARM-Linux. I due di maggiore interesse per questo lavoro di tesi sono: *linux-arm.com*[5] e *imxdev.org*[6], costruiti con il paradigma wiki per il quale ognuno può contribuire all'aggiornamento dell'informazione presentata, il primo documenta il progetto ARM-Linux senza entrare nelle specificità delle diverse piattaforme, mentre il secondo pone maggiore fuoco sulla gestione e utilizzo di kernel compilati per la famiglia di processori Freescale i.MX.

6.1.1 Mailing list e gestione delle Patch

Le mailing list sono lo strumento di aggregazione preferito dalla comunità di sviluppatori Linux[7]. Ogni gruppo di sviluppo possiede in genere al-

[4] `http://www.arm.linux.org.uk`

[5] `http://www.linux-arm.com`

[6] `http://www.imxdev.org/`

[7] Un elenco completo di tutte le mailing list relative a progetti inerenti al kernel Linux può essere reperito al link: `http://vger.kernel.org/vger-lists.html`

meno due mailing list: una dedicata agli utenti dove è possibile discutere solamente di aspetti legati all'utilizzo del codice prodotto ed una dedicata agli sviluppatori (dev) utilizzata in ogni fase di intervento nel codice stesso (proposte di modifica, review, richiesta di spiegazioni specifiche riguardo il codice, discussioni in merito a possibili evoluzioni).

Ciò che rende questo strumento particolarmente adatto alla comunità di sviluppatori è l'integrazione creata tra Mail Client molto diffusi come *Mutt*[8] e *Pine*[9] ed il software di version management *Git*[10], capace di automatizzare gran parte degli aspetti che riguardano la gestione dell'evoluzione di un progetto sviluppato in modo distribuito.

Importanti sono gli archivi delle suddette mailing list, nei quali è possibile trovare tutte le discussioni già affrontate nel corso del tempo, da considerarsi spesso come una delle poche di documentazione per il progetto.

Il progetto ARM-Linux possiede quattro mailing list[11] delle quali tre sono attive:

linux-arm mailing list per gli utenti, utilizzata per discussioni generali.

linux-arm-toolchain utilizzata per lo scambio di informazioni riguardo la generazione di una toolchain adatta.

linux-arm-kernel mailing list per gli sviluppatori, vero centro di aggregazione della comunità ARM-Linux.

linux-arm-kernel è la mailing list utilizzata per l'invio ed il review delle patch:

> Una *patch* è un file testuale standardizzato dove vengono descritte le modifiche proposte al codice sorgente del progetto; generato automaticamente da utility apposite (principalmente basate sul programma diff), è formattato in modo tale da poterne applicare le modifiche ad un repository locale in modo automatico.

Le patch inviate alla mailing list *linux-arm-kernel* devono sottostare ad alcune condizioni fondamentali per poter essere accettate[42]:

1. Contenuto della patch:

[8] `http://www.mutt.org/`
[9] `http://www.washington.edu/pine/`
[10] `http://git-scm.com/`
[11] `http://www.arm.linux.org.uk/mailinglists/lists.php`

- Dovrebbe coprire una funzionalità oppure un singolo driver oppure un singolo bug-fix, modifiche a file di sistema dovrebbero essere isolate in patch separate.
- Ogni patch presentata, se applicata, non dovrebbe inreferire nella compilabilità del kernel.
- Il contenuto della patch deve essere consistente.
  File patch validi possono essere creati con il comando `git format-patch` dalla base-directory del kernel tree dopo aver confermato le modifiche con il comando `git commit`.
- Il codice della patch deve sottostare alle direttive Coding Style[10] del kernel Linux ed essere preservato da possibili variazioni (sostituzione del carattere tab, wrap automatico delle linee).

2. L'oggetto della patch deve essere significativo:
   - Deve contenere i tag specifici che indicano a quale piattaforma e board ci si sta riferendo (per la piattaforma: MXC, PXA, OMAP .. per la board: pcm037, sa1100 , Armadillo5x0..).
   - Deve contenere un indicativo di versione della patch (se riproposta con modifiche).
   - Deve descrivere brevemente la modifica apportata.
3. Il corpo della mail deve iniziare con una descrizione dettagliata delle modifiche apportate.
4. Patch che richiedono una catena di dipendenze devono esplicitamente definirla.
5. Deve essere contenuta l'indicazione di branch e versione del kernel sulla quale è stata basata la patch.
6. Deve essere presente la firma dell'autore con il tag *Signed-off-by*:
7. E' preferibile che il contenuto della patch sia incollato nel corpo della mail.

Patch incomplete o formalmente errate non proseguono nel processo di review.

I membri della comunità ARM-Linux sono riuniti in centri di interesse che gravitano attorno alle diverse piattaforme. Ogni gruppo gestisce un proprio repository di sviluppo contenente le patch recentemente approvate per tale piattaforma; periodicamente i mantainer della piattaforma provvedono a sincronizzare lo sviluppo con il repository principale ARM-Linux (detto rmk gestito da Russell King) anche questo periodicamente

sincronizzato con il repository mainline di Linus con cadenze costanti secondo il processo di sviluppo standard del kernel [12].

Il gruppo che mantiene la piattaforma per processori freescale i.MX è fortemente supportato dall'azienda Tedesca Pengutronix[13] che ne gestisce essa stessa il repository Git pubblico di sviluppo[14].

Per fare in modo che le patch proposte attraverso la mailing list *linux-arm-kernel* vengano notate velocemente dai componenti di questo gruppo, l'oggetto della mail deve contenere uno dei due tag IMX o MXC.

6.1.2 Organizzazione del codice di supporto per l'architettura ARM

Dalla versione 2.6.27 del kernel Linux è stato eseguito un riposizionamento di tutto il codice specifico ARM. Ora i sorgenti specifici per questa architettura (compresi i file header) risiedono interamente nella sottodirectory `arch/arm` del kernel tree, organizzati nella seguente struttura:

kernel - contiene le funzionalità base del kernel specifiche per l'architettura ARM, assieme alle routine di inizializzazione.

mm - contiene il codice di gestione della memoria (tlb, mmu, cache, dma).

lib - contiene librerie specifiche e/o ottimizzate per l'architettura ARM come: backtrace, memcpy, funzioni di i/o, div etc...

include - contiene i file header per l'architettura ARM comuni a tutte le piattaforme.

nwfpe e vfp - contengono implementazioni software e hardware delle librerie di calcolo in virgola mobile.

common - contiene alcuni sorgenti di supporto tra cui, routine di controllo per VIC, implementazione del sistema *clockdev* e supporti a piattaforme hardware comuni.

ptat-xx - directory contenenti sorgenti e file header di supporto per famiglie di System on Chip (mxc, omap, orion, pxa ..).

[12] Si consiglia la lettura della documentazione in tree del kernel a questo proposito [11].
[13] `http://www.pengutronix.de`
[14] Branch mxc-master in `git://git.pengutronix.de/git/imx/linux-2.6.git`

mach-xx -directory contenenti il codice specifico a supporto delle singole board raggruppate per specifici System on Chip.

boot - directory che conterrà il file immagine del kernel compilato.

tools - contiene script per generare file come *mach-types.h*.

oprofile - contiene librerie utili per il profiling low level del sistema.

configs - contiene i file di configurazione predefiniti per ogni board arm-based.

Una piattaforma di supporto per una determinata board ARM-based è composta dal codice contenuto nella directory *plat-ArmCpuFamily* (con ArmCpuFamily famiglia del processore utilizzato nella board) e dai sorgenti specifici contenuti in *mach-CpuType* (con CpuType tipologia di processore utilizzata).

6.1.3 Fasi di boot di Linux su architettura ARM

Dai compiti del Bootloader all'esecuzione del processo *init*, qui è presentata nel dettaglio la sequenza delle operazioni che portano all'inizializzazione dell'ambiente Linux sull'architettura ARM esplicitandone i contenuti per la famiglia di processori i.MX.

Il boot loader

Il boot loader è un piccolo programma che esegue prima di Linux all'avvio del sistema, il cui compito è quello di effettuare alcune inizializzazioni preliminari dell'hardware ed alla fine chiamare la routine iniziale di Linux. Esistono diversi boot loader che supportano Linux su piattaforma ARM (U-Boot[15], Blob[16], Redboot[17]).

Dalla documentazione presente nella pagina web del progetto ARM-Linux[44, 46] si apprendono le operazioni di base che Linux si aspetta vengano eseguite dal boot loader:

Setup e inizializzazione della RAM. Il boot loader deve cercare ed inizializzare tutti i dispositivi di memoria RAM utilizzabili dal kernel come memoria volatile.

[15] http://www.denx.de/wiki/U-Boot
[16] http://sourceforge.net/projects/blob/
[17] http://sources.redhat.com/redboot/

La configurazione della memoria trovata deve essere comunicata al kernel attraverso parametri `ATAG_MEM`. La memoria RAM non deve essere necessariamente contigua (per indirizzi fisici), multipli parametri `ATAG_MEM` indicano differenti blocchi di memoria RAM non contigui.

Inizializzazione di una porta seriale. La porta seriale inizializzata verrà utilizzata dal kernel per i primi messaggi di debug, antecedenti l'inizializzazione del vero e proprio driver seriale Linux.

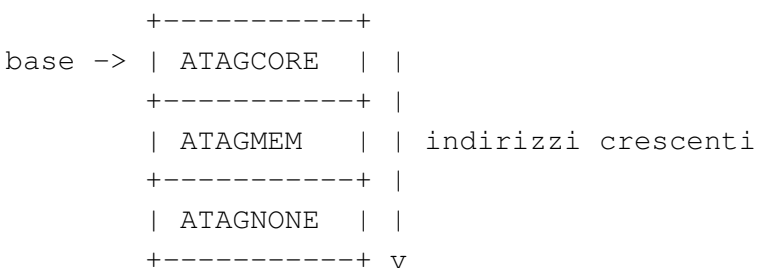
Le stringhe di debug successive verranno scritte nella console di sistema (che può essere diretta sulla stessa porta seriale oppure a video) definita dal parametro del kernel `console=...` (Per una descrizione completa dei parametri possibili si faccia riferimento al documento [8]).

Riconoscimento della macchina. Il boot loader deve riconoscere il tipo di macchina sul quale sta eseguendo e passarne l'id associato al kernel. E' attraverso l'id di macchina che il kernel individuerà la piattaforma corretta da utilizzare.

La lista degli id supportati dalla versione locale dei sorgenti è presente nel file *linux/arch/arm/tools/mach-types* mentre in rete [18] è presente la versione aggiornata.

Inizializzazione della kernel tagged list. Il boot loader deve creare ed inizializzare la lista dei tag utilizzata per passare i parametri del kernel.

E' una struttura dati standardizzata dove i parametri vengono delimitati da tag di separazione. Una lista valida inizia con `ATAG_CORE` e termina con `ATAG_NONE`. Una lista minimale per il kernel potrebbe essere la seguente:



[18] `http://www.arm.linux.org.uk/developer/machines/`

La lista deve risiedere in RAM in uno spazio di memoria che non deve essere compromesso dalle successive attività del kernel (decompressione o inizializzazione del RAM disk); solitamente risiede nei primi 16KB della memoria di sistema.

Esecuzione del kernel. Esistono due modalità di esecuzione del kernel Linux, la prima (normale) prevede la copia dell'intera immagine in RAM in una posizione appropriata (il kernel si riserva di utilizzare i 16 KB inferiori all'indirizzo di start che sommati allo spazio per la tagged list colloca solitamente l'immagine a 32KB dall'inizio della RAM).

La seconda (XIP, eXecituin In Place) permette a kernel compilati appositamente di eseguire direttamente dal dispositivo di memoria di massa dove risiede.

Ad ogni modo in entrambe le modalità l'ambiente di esecuzione deve essere così configurato:

- Registri della CPU
  - `r0` = 0.
  - `r1` = id di macchina.
  - `r2` = indirizzo fisico in RAM della tagged list (solitamente con offset 0x100 rispetto all'inizio della RAM).
- Modalità CPU
  - Tutti gli interrupt devono essere disabilitati.
  - La CPU deve essere in modalità SVC (Supervisor Call).
- Cache, MMU
  - Il dispositivo MMU non deve essere attivo.
  - La cache istruzioni può essere attiva.
  - La cache dati deve essere disabilitata e vuota.
- Dispositivi
  - Non devono essere attivi trasferimenti DMA da e per dispositivi.
- Il boot loader infine deve saltare alla prima istruzione dell'immagine del kernel (inizio dell'immagine).

E' il kernel ora ad avere il controllo della CPU. Di seguito, passo per passo, sono riportate le fasi di inizializzazione dell'ambiente Linux a partire da un'immagine compressa, ottenute analizzando il codice sorgente del kernel in riferimento alla traccia fornita dal documento [26].

Decompressione dell'immagine

- Il boot loader salta alla label **`start`** in *arch/arm/boot/compressed/head.S.*
- I parametri passati in `r1` (id macchina) e `r2` (indirizzo della atag list) sono salvati.
- Disabilita gli interrupt ed aggiorna gli indirizzi rispetto all'offset di esecuzione.
- Abilita la cache dati chiamando la procedura **`cache_on`**.
  All'interno di **`cache_on`** viene individuata l'architettura di sistema nella lista `proc_types`.
- Inizializza gli indirizzi per la chiamata alla routine di decompressione:
  `r4` = indirizzo fisico di inizio kernel, `sp` = indirizzo della routine di decompressione.
- Controlla che l'immagine decompressa non sovrascriverà l'immagine zImage.
- Chiama la routine di decompressione **`decompress_kernel()`** (presente nel file *arch/arm/boot/compressed/misc.c*).
  `decompress_kernel()` stamperà il messaggio `"UncompressingLinux..."` nel terminale di output, proseguendo nella chiamata della funzione **`gunzip()`** ed in seguito alla visualizzazione del messaggio `"done, booting the kernel"`.
- Pulisce e disabilita la cache dati per ricreare le condizioni per la routine **`start`** del kernel.
- Salta all'inizio dell'immagine decompressa del kernel, indirizzo salvato nel registro `r4`.
  Questo indirizzo è specifico per ogni piattaforma, definito nella variabile di sistema zreladdr. Per la piattaforma i.MX questa variabile è definita nel file *arch/arm/mach-mx3/Makefile.boot* (`zreladdr-y := 0x80008000`).

Codice kernel specifico del processore ARM

Dopo la decompressione del kernel, la routine eseguita è dipendente dall'architettura. Nel caso di processori ARM all'indirizzo `zreladdr` è pre-

sente la procedura **stext** dal file *arch/arm/kernel/head.S* che provvede a:

- Passare alla modalità Supervisore e disabilitare gli interrupt.
- Individua il tipo di processore attraverso la procedura **__lookup_processor_type** definita in *arch/arm/kernel/head-common.S*.
  Questa ritornerà un puntatore ad una struttura `proc_info_list` definita nel file *arch/arm/include/asm/procinfo.h* che contiene puntatori a routine specifiche per l'architettura individuata (*arch/arm/mm/proc-v6.S*: **__v6_proc_info**).
- Individua il tipo di macchina attraverso l'id salvato con la procedura **__lookup_machine_type** definita nel file *arch/arm/kernel/head-common.S*.
  Questa ritornerà un puntatore ad una struttura di tipo `machine_desc` definita nel file *arch/arm/include/asm/mach/arch.h* ed inizializzata nel sorgente specifico nella cartella *arch/arm/mach-mx3/*.
- Crea la page table attraverso la procedura **__create_page_tables** abbastanza grande per mappare almeno il codice del kernel.
- Salta a `*(__v6_proc_info + #PROCINFO_INITFUNC)` che risulta nella chiamata a **__v6_setup** nel file *arch/arm/mm/proc-v6.S*. Inizializzando così la TLB, la cache ed il modulo MMU.
- Abilita il modulo MMU con **__enable_mmu**, che dopo alcune operazioni preliminari chiamerà **__turn_mmu_on** (indirizzo salvato in `lr` ancora in **stext** *arch/arm/kernel/head.S*).
- Da **__turn_mmu_on**, dopo operazioni nei registri di configurazione, viene chiamata **__switch_data** (indirizzo salvato in `r13` ancora in **stext**) che eseguirà **__mmap_switched** (*arch/arm/kernel/head-common.S*).
- In **__mmap_switched**:
  - Viene copiato il segmento dati in RAM.
  - il segmento BSS viene cancellato.
  - Al termine viene chiamata la routine **start_kernel()** nel file *init/main.c*.

Ora l'ambiente è pronto per la porzione del kernel indipendente dall'architettura.

Il kernel vero e proprio

In **`start_kernel()`**: (*`init/main.c`*)

- Disabilita gli interrupt per la prima CPU con **`local_irq_disable()`** (*`include/linux/irqflags.h`*).
- Acquisisce il lock del kernel con **`lock_kernel()`** per esser sicuri che l'esecuzione non venga interrotta o subisca preemption da interrupt a priorità maggiori (*`lib/kernel_lock.c`*).
- Registra la routine di notifica dei tick con **`tick_init()`** (*`/kernel/time/tick-common.c`*).
- Attiva il primo processore (CPU0) con **`boot_cpu_init()`** (*`init/main.c`*).
- Inizializza il sottosistema di gestione della memoria con **`page_address_init()`** (*`mm/highmem.c`*).
- Visualizza la versione del kernel nella console con **`printk(linux_banner)`** (*`init/version.c`*).
  A questo punto ancora non è attiva nessuna console, i messaggi vengono scritti da *`printk()`* in un buffer di memoria e solo quando verrà caricata la console di sistema i messaggi potranno essere letti dall'utente.
- Configura l'hardware specifico come memorie, dispositivi di I/O etc con **`setup_arch(&command_line)`**.
  Il parametro `command_line` è la tagged list passata dal boot loader che contiene i parametri del kernel. (*`arch/arm/kernel/setup.c`*).
  - In **`setup_arch(&command_line)`**:
    Vengono inizializzati i processori della board con **`setup_processor()`** che salva le informazioni in cache, inizializza le informazioni specifiche SMP e inizializza gli stack specifici per-cpu (*`arch/arm/kernel/setup.c`*).
    `cpu_proc_init()` è una macro dipendente dall'architettura che per l'architettura CPU_V6 punta a **`cpu_v6_proc_init()`** in *`arch/arm/mm/proc-v6.S`* che non fa nulla.
  - La chiamata a **`setup_machine(machine_arch_type)`** ritorna la struttura `machine_desc` propria della macchina individuata, definita nel file *`arch/arm/include/asm/mach/arch.h`* ed inizializzata nel sorgente specifico nella cartella *`arch/arm/mach-mx3/`*.

  - Localizza e converte il formato (se necessario) della lista dei parametri del kernel.
  - Esegue **`mdesc->fixup()`** per terminare operazioni pendenti di inizializzazione dell'hardware dipendenti dalla piattaforma.
  - In **`paging_init(mdesc)`** (*arch/arm/mm/mmu.c*) configura la page table con le tabelle *zero page* e *bad page* ed inizializza le mappature statiche in memoria virtuale con **`mdesc->map_io()`** in **`devicemaps_init(mdesc)`**.
  - **`request_standard_resources(&meminfo, mdesc)`** dove `mdesc` è utilizzata per inizializzare la memoria video (se è definito `mdesc->video_start`) e le risorse LPT (lp0,lp1,lp2).
  - **`cpu_init()`** completa l'inizializzazione della CPU.
  - Inizializza i puntatori alle funzioni dipendenti dall'hardware su cui è in esecuzione:
    ```
    init_arch_irq = mdesc->init_irq;
    system_timer = mdesc->timer;
    init_machine = mdesc->init_machine;
    ```
  - Esegue **`early_trap_init()`**.
- Configura lo scheduler di Linux con **`sched_init()`** (*kernel/sched.c*)
  - Inizializza la coda dei processi.
  - Crea un thread idle con **`init_idle(current, smp_processor_id())`** (*kernel/sched.c*).
- Inizializza le zone di memoria come *DMA*, *normal*, *high memory* con **`build_all_zonelists()`** (*mm/page_alloc.c*).
- Processa i parametri del kernel con **`parse_early_param()`** (*init/main.c*) e **`parse_args()`** (*kernel/params.c*).
- Inizializza la tabella degli interrupt, il Generic Interrupt Controller ed il vettore di cattura delle eccezioni con **`init_IRQ()`** (*arch/arm/kernel/irq.c*) e **`trap_init()`** (*arch/arm/kernel/traps.c*).
  Assegna inoltre le affinità per gli interrupt ai differenti processori.
- Prepara la CPU di boot ad accettare le notifiche dei tasklets con **`softirq_init()`** (*kernel/softirq.c*).
- Inizializza e fa partire il timer di sistema con **`time_init()`** (*arch/arm/kernel/time.c*).

- Abilita gli interrupt locali alla CPU di boot con **`local_irq_enable()`** (*include/linux/irqflags.h*).
- Inizializza la console di sistema con **`console_init()`** (*drivers/char/tty_io.c*).
- Trova il numero totale di pagine libere in tutte le zone di memoria con **`mem_init()`** (*arch/arm/mm/init.c*).
- Inizializza il gestore della memoria slab[19] con **`kmem_cache_init()`** (*mm/slab.c*).
- Determina la velocità del processore in BogoMips con **`calibrate_delay()`** (*init/calibrate.c*).
- Inizializza i componenti interni del kernel come *SLAB caches*, *VFS*, *buffer*, *code di segnali*, *massimo numero di thread e processi*, etc...
- Inizializza il filesystem *proc/* con **`proc_root_init()`** (*fs/proc/root.c*).
- Chiama **`rest_init()`** che creerà il processo con ID 1.
  - In **`rest_init()`** (*init/main.c*): il kernel è "vivo" non è una funzione marcata con il modificatore `__init` e quindi non sarà memoria liberata successivamente perché inutile.
  - Continua la sequenza di inizializzazione che andrà a creare il processo `init` con **`kernel_thread(kernel_init, NULL, CLONE_FS | CLONE_SIGHAND)`**.
  - Crea il thread demone del kernel, radice di generazione di ogni kernel thread, con **`kernel_thread(kthreadd, NULL, CLONE_FS | CLONE_FILES) (kernel/kthread.c)`**.
  - Libera il lock del kernel acquisiti all'inizio di **`start_kernel()`** con **`unlock_kernel()`** (*include/linux/smp-lock.h*).
  - Esegue la routine **`schedule()`** (*kernel/sched.c*).
    Ora l'ambiente di esecuzione è multitasking.
  - Esegue la funzione **`cpu_idle()`** (*arch/arm/kernel/process.c*).
    Questo thread è il codice in esecuzione quando nessun altro processo nel sistema è attivo.
    Mantenere la CPU attiva con un thread che non fa nulla serve a risparmiare energia e mantenere basse le latenze del sistema.

[19] Memory allocation subsystem `http://www.ibm.com/developerworks/linux/library/l-linux-slab-allocator/?ca=dgr-lnxw07LinuxSlabAllo`

In **kernel_init()**: (*init/main.c*)

- Inizia a preparare l'ambiente SMP con **smp_prepare_cpus()** (*arch/arm/mach-realview/platsmp.c*)che non fa nulla per architetture UP .
- Configura alcuni parametri dello scheduler con **sched_init_smp()** (*kernel/sched.c*).
- In **do_basic_setup()** inizializza il supporto alle work queue con **init_workqueues()**, il driver model con **driver_init()** (*drivers/base/init.c*) ed altre inizializzazioni.
- **do_initcalls()**.
- Infine chiama **init_post()** (*init/main.c*) dove si entra nella modalità utente cercando di creare il processo di inizializzazione tentando in sequenza:
  - **run_init_process("/sbin/init")**
  - **run_init_process("/etc/init")**
  - **run_init_process("/bin/init")**
  - **run_init_process("/bin/sh")**

Il processo `init` cerca di inizializzare l'ambiente user del sistema per poi trasferire il controllo alla console e rimanere in esecuzione.

6.1.4 Metodi di debug

Linus Torvalds è categorico nella sua mail a riguardo[47]: Linux non contiene e non vorrà mai contenere un modulo *kernel debugger* che permetta l'esecuzione step-by-step delle istruzioni alla ricerca della sola che produce il bug.

Le motivazioni sono nobili, mirano ad una maggiore conoscenza e consapevolezza dei meccanismi interni del kernel da parte degli sviluppatori rispetto alla mera analisi delle interazioni tra righe di codice.

Il risultato di questa linea di pensiero è che l'unica via software possibile per tracciare l'esecuzione del kernel è l'output nella console di sistema, ottenuto tramite chiamate alla funzione **printk()**[20] oppure bug trace di eventi *oops* o *kernel panic*.

[20]Esistono dei casi in cui la console è un lusso che lo sviluppatore non può permettersi, principalmente nello sviluppo di un nuovo hardware abstraction layer quando ancora non è possibile instaurare una connessione seriale, allora attraverso hardware debug in-

6.1.4.1 Early debug

> Con *Early debug* si intende il tracciamento dell'esecuzione del kernel tramite stringhe di output nella prima fase di boot dove ancora non è stata caricata la console di sistema ed il canale di comunicazione tra macchina host (di analisi) e macchina target (dove sta eseguendo la versione di test del kernel) è la sola connessione seriale (standard RS232).

Come descritto nella sezione 6.1.3 le prime stringhe di debug vengono scritte direttamente nella porta seriale se correttamente configurata dal boot loader. Questo è possibile attraverso la procedura assembly **`printascii`** (*`arch/arm/kernel/debug.S`*) ed alle procedure di debug low level abilitate nella configurazione dal parametro `CONFIG_DEBUG_LL`.

Dal messaggio `"done, booting the kernel"` però, ogni messaggio di log segue la strada della console attraverso la funzione **`printk()`** anche se una console vera e propria viene caricata molto più tardi nel processo di boot oscurando così i messaggi di debug di questa fase nel caso avvenisse una condizione di blocco del kernel.

La soluzione proposta da Russell King[43] è inserire una chiamata alla procedura **`printascii`** all'interno della funzione **`printk()`** che stampi lo stesso messaggio che **`printk()`** sta salvando in memoria direttamente nella porta seriale, rendendo così possibile la lettura del reale flusso di log.

Una patch che inserisce questa modifica è la seguente:

Listing 6.1: Patch ARM Make low-level printk work

```
From 0c61b75f9da1a0889959a0f9bd0b8b63f936ddf3 Mon Sep 17 00:00:00 2001
From: Tony Lindgren <tony@atomide.com>
Date: Mon, 9 May 2005 14:10:26 -0700
Subject: [PATCH] ARM: Make low-level printk work

Makes low-level printk work.

Signed-off-by: Tony Lindgren <tony@atomide.com>
---
 kernel/printk.c |    8 ++++++++
 1 files changed, 8 insertions(+), 0 deletions(-)

diff --git a/kernel/printk.c b/kernel/printk.c
index e3602d0..e39866e 100644
```

terface, come BDI3000, è possibile avere il controllo completo del processore utilizzando l'interfaccia Jtag. I metodi di debug hardware non verranno trattati in questo documento.

```
--- a/kernel/printk.c
+++ b/kernel/printk.c
@@ -44,6 +44,10 @@ void asmlinkage __attribute__((weak)) early_printk(const char
     *fmt, ...)

 #define __LOG_BUF_LEN  (1 << CONFIG_LOG_BUF_SHIFT)

+#ifdef        CONFIG_DEBUG_LL
+extern void printascii(char *);
+#endif
+
 /* printk's without a loglevel use this.. */
 #define DEFAULT_MESSAGE_LOGLEVEL 4 /* KERN_WARNING */

@@ -668,6 +672,10 @@ asmlinkage int vprintk(const char *fmt, va_list args)
                                sizeof(printk_buf) - printed_len, fmt, args);


+#ifdef CONFIG_DEBUG_LL
+       printascii(printk_buf);
+#endif
+
        /*
         * Copy the output into log_buf.  If the caller didn't provide
         * appropriate log level tags, we insert them here
--
1.5.4.3
```

Attenzione al conflitto di scrittura sulla porta seriale che avviene successivamente al corretto caricamento della console di sistema (se questa scrive sulla porta seriale) tra le due procedure di output dei log del kernel. Chiaramente dovrà essere disabilitata l'opzione `CONFIG_DEBUG_LL` per utilizzare in modo univoco il driver seriale caricato appositamente.

6.1.4.2 Normal Debug

Successivamente al caricamento della console di sistema questa provvederà a mostrare tutti i messaggi in coda nel buffer dedicato. Come detto in precedenza per scrivere su tale buffer viene utilizzata la funzione **`printk()`**.

`printk()` si comporta come la normale funzione **`printf()`** nel parsing delle stringhe ma prevede un parametro obbligatorio di `debug-level` da inserire all'inizio del formato della stringa per filtrare i messaggi in console a seconda delle preferenze espresse dall'utente (parametro `loglevel=...` del kernel).

Allora la chiamata a **`printk()`** deve essere nella forma **`printk(LEVEL fmt, ...)`** ed i livelli sono:

KERN_EMERG	$''<0>''$	Comunicazioni nel momento in cui il sistema è inutilizzabile.
KERN_ALERT	$''<1>''$	Azioni che devono essere fatte immediatamente.
KERN_CRIT	$''<2>''$	Condizioni critiche.
KERN_ERR	$''<3>''$	Errori di sistema.
KERN_WARNING	$''<4>''$	Messaggi di attenzione.
KERN_NOTICE	$''<5>''$	Messaggi significativi normali.
KERN_INFO	$''<6>''$	Messaggi informativi (livello di default, vengono stampati tutti i messaggi considerati più seri di quelli di debug).
KERN_DEBUG	$''<7>''$	Messaggi utilizzati solo per debug. Solitamente viene utilizzata la macro wrapper **`pr_debug(fmt, ...)`** che si risolve in **`printk(KERN_DEBUG fmt, ...)`** se è definito il parametro di configurazione `DEBUG`.

per utilizzare **`printk()`** nel codice del kernel è necessario includere il file header *`linux/kernel.h`*.

6.1.4.3 Driver Debug

Le funzioni di log consigliate per il debug di driver sono **`dev_emerg()`**, **`dev_alert()`**, **`dev_crit()`**, **`dev_err()`**, **`dev_warn()`**, **`dev_notice()`**, **`dev_info()`** e **`dev_debug()`** wrapper tutte della funzione **`dev_printk(LEVEL , dev , format , ...)`** che prevede in ingresso il parametro `dev` contenente le informazioni del device per il quale si sta scrivendo il messaggio di log.

`dev_printk()` utilizza infine la normale `printk()` per scrivere nella console risultando in un messaggio nella forma:

```
devicename: message
```

Tutte queste funzioni vengono abilitate dal parametro di configurazione `DEBUG`.

6.2 Ambiente di sviluppo, la macchina host

> La macchina *host* o *workstation* è quel Personal Computer utilizzato per lo sviluppo del software che eseguità nella macchina *target*. Solitamente contiene tutti i tool necessari ad acquisire il codice sorgente, modificalo, compilarlo e trasferirne i file binari nella macchina target.

Di seguito sono riportate le caratteristiche importanti della macchina host utilizzata:

CPU:	Intel(R) Pentium(R) M processor 1.73GHz cache size : 2048 KB bogomips : 1597.75
Memoria RAM:	2074640 KB
Sistema Operativo:	Ubuntu Linux 8.04 "Hardy Heron" rilasciato nell'aprile 2008.

Importante è la configurazione della shell bash per la gestione dei meccanismi di compilazione e per automatizzare determinate attività.

Per questo si è scelto di creare uno script di inizializzazione chiamato *android.sh* che verrà via via riempito con le configurazioni di ambiente necessarie, da eseguire nella shell corrente qualora si intendesse utilizzarla per lo sviluppo:

```
$ mkdir ~/bin
$ touch ~/bin/android.sh
$ chmod +x ~/bin/android.sh
```

Il file *android.sh* inizialmente verrà definito così:

Listing 6.2: *android.sh* Forma iniziale.

```
#!/bin/bash
#Directory dove risiedono tool eseguibili privati
export PATH=$PATH:$HOME/bin

#Shell di lavoro
bash
```

Aprendo una shell, per inizializzare l'ambiente di lavoro, basterà eseguire:

```
$ ./bin/android.sh
```

Note sugli script per shell:
`#!/bin/bash` deve essere sempre la prima riga di uno script shell.

Le variabili di ambiente sono definite in una shell Linux in questo modo:

```
$ VARIABILE=Valore
```

e vengono richiamate con il modificatore $:

```
$ echo $VARIABILE
  Valore
```

Per creare una nuova variabile in una sola riga ${}:

```
$ echo ${FRANKY:=Franky}
  Franky
```

Mentre per valutare espressioni aritmetiche $[ ]:

```
$ echo $[${FOUR=4}+2]
  6
```

La keyword export invece ampia la visibilità della variabile esportata (altrimenti locale alla shell di creazione) a tutti i processi figli della presente shell.

6.2.1 Posizione del progetto

La directory root del progetto deve essere accessibile in lettura e scrittura e risiedere in una partizione abbastanza capiente da contenere tutti i sorgenti ed extra tool necessari. Allora:

```
$ mkdir /media/dati/android
```

e viene inserita la riga seguente all'interno di *android.sh*:

Listing 6.3: *android.sh* La variabile PRJROOT.

```
+export PRJROOT=/media/dati/android

#Shell di lavoro
```

Nel proseguo della trattazione ci si riferirà a tale directory con la variabile d'ambiente `PRJROOT`.

La directory radice del progetto acquisirà via via la seguente struttura:

build-tools : Conterrà tutto ciò che è necessario a creare la cross-toolchain.

build-BSP : Conterrà tutto ciò che è necessario a creare un root filesystem valido.

kernelLinus : Repository locale del mainline kernel.

kernelMXC : Repository locale del branch MXC.

rootfs : Filesystem root per la macchina target.

tools : Conterrà la completa toolchain cross-platform.

6.2.2 Il kernel

Git è il software di version control distribuito utilizzato per mantenere i repository di sviluppo del kernel.

La documentazione completa dei comandi git risiede nel sito ufficiale [21] mentre una versione più snella è presentata nel documento [31].

Git può essere installato e configurato con:

```
$ sudo apt-get install git-core
$ git config --global user.name "Alberto Panizzo"
$ git config --global user.email maramaopercheseimorto@gmail.com
```

Successivamente si procede alla clonazione del repository del mainline kernel:

```
$ mkdir $PRJROOT/kernelLinus
$ cd $PRJROOT/kernelLinus
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/
   torvalds/linux-2.6.git
```

Così in *$PRJROOT/kernelLinus/linux-2.6/* è presente una versione completa del mainline kernel all'ultimo stadio di evoluzione.

Per sincronizzare il repository locale con le ultime modifiche remote:

```
$ cd $PRJROOT/kernelLinus/linux-2.6/
$ git pull
```

[21] `http://git-scm.com/documentation`

Dato che il branch di sviluppo per gli Application Processor i.MX è ancora oggi in continua evoluzione, per sfruttare le ultime funzionalità senza attendere il lungo processo di merge nel mailine branch[22] ne è utile la clonazione in locale:

```
$ mkdir $PRJROOT/kernelMXC
$ cd $PRJROOT/kernelMXC
$ git clone git://git.pengutronix.de/git/imx/linux-2.6.git
$ git checkout origin/mxc-master
```

L'ultima riga di comando indica a git di spostarsi nel branch corretto: *mxc-master* del repository Pengutronix.

6.2.3 Rudimenti di Git per la creazione di patch

Un repository Git contiene al suo interno, oltre ai file sorgenti del progetto, dei metadati registrati in un database didtribuito detto *index* utilizzati da Git per rilevare le ultime modifiche apportate dopo l'ultima *commit*. Innanzitutto:

> Effettuare una *commit* significa registrare nel database index le ultime modifiche apportate al repository locale successive alla precedente commit tramite il comando `git commit`.

Ad una commit viene assegnato un identificativo hash univoco che la colloca in un punto preciso della storia del repository git: questo riflette il fatto che le modifiche registrate in una commit acquisiscono un senso solo se considerate come ultime di un preciso stack di altre commit che determinano l'evoluzione locale del progetto.

Una commit solitamente è descritta da un messaggio breve (espresso nell'opzione `-m`) che ne indica l'argomento toccato e da una spiegazione più lunga (esprimibile con l'opzione `-F-`) che ne sviluppa le motivazioni e gli interventi veri e propri della modifica.

Una commit può essere firmata dal suo autore (deve esserlo nel caso si volesse procedere al review della patch associata) con l'opzione `-s` se l'ambiente git è correttamente configurato.

[22] Attenzione! Il processo di review delle patch alla piattaforma i.MX avviene a questo livello. Il codice risultante procede nella catena di merge nei repository gerarchicamente più elevati senza nessuna modifica.

Git offre dei comandi utili a valutare lo stato del repository locale e a decidere quali delle ultime modifiche non registrate debbano andare a comporre la successiva commit:

`git status` : Riporta lo stato del repository locale mostrando quali file sono stati modificati e quali sono stati aggiunti e non considerati dall'indice git.

`git add` : Aggiunge le modifiche ad un determinato file nell'insieme di quelle che andranno a formare la prossima commit. Attenzione che, se lo stesso file verrà modificato successivamente la registrazione con `git add`, le ultime modifiche non verranno considerate nella commit.

`git diff` : Serve a visualizzare le ultime modifiche effettuate. Se richiamato senza opzioni riporta le modifiche ai file del repository locale effettuate successivamente l'ultima commit e successivamente a quelle temporaneamente registrate con `git add`. Se richiamato con il parametro `- -cached`, visualizza solo le modifiche registrate temporaneamente con `git add` che andranno a comporre la prossima commit.

Alle volte è necessario annullare una o più delle recenti commit, a questo proposito esiste il comando `git reset <commit>` che presenta tre diverse modalità : la prima di default è esplicitata dall'opzione `- -mixed`, le informazioni di commit successive a quella selezionata vengono eliminate dal database index, le modifiche relative sono mantenute nel repository e considerate come nuove. La seconda viene definita con l'opzione `- -soft` che a differenza della prima, oltre a cancellare le informazioni di indice considera le modifiche relative come temporaneamente registrate per la commit successiva. La terza `- -hard` resetta sia le informazioni nel database, sia le modifiche ai file per avere un repository pulito alla commit selezionata.

Le commit in questo comando vengono identificate nella forma:

`HEAD` : per l'ultima commit eseguita, `git reset HEAD` azzera le informazioni di modifiche temporaneamente registrate per la commit successiva, `git reset - -hard HEAD` annulla tutte le ultime modifiche apportate al repository.

`HEAD^`: è la commit precedente all'ultima registrata.

`HEAD~n`: riferisce l'n-esima commit precedente l'ultima registrata.

Ad ogni commit può essere associata una patch da sottoporre al processo di review attraverso le mailing list. Per questo esiste il comando `git format-patch` che fornisce un metodo completo per creare file di patch validi per la serie di commit eseguite a partire da una in particolare oppure per un range definito di commit. L'output di format-patch è regolato dalle sue opzioni, delle quali le seguenti sono normalmente utili:

`-o dir` per definire la directory di output dei file di patch prodotti.

`-n` inserisce come prefisso al messaggio di commit la stringa [PATCH n/m] con n numero attuale del gruppo ed m numero totale delle patch del gruppo.

`- -check` esegue dei controlli formali sullo stile del codice come spazi bianchi oltre la fine delle righe e spazi prima di tab nelle indentazioni (Codice scritto con errori in questo senso non viene neppure considerato nel processo di review).

L'ultimo parametro indica generalmente da che punto di commit generare le patch. Se indicato `origin` verranno create le patch di tutte le commit effettuate dalla creazione del repository locale. Se indicato un numero ad es. `-3` verranno create le patch solamente delle ultime 3 commit.

Tutti i comandi qui presentati hanno una pagina di descrizione approfondita in `man` accessibile con `man git-nome-comando`.

6.2.4 Cross-Platform Toolchain

Una Toolchain (catena di attrezzi) è quell'insieme di programmi necessari a creare i file eseguibili di un software. Normalmente sono inclusi un linker, assembler, archiver, compiler, librerie e file header per il dato linguaggio di programmazione.

Una Cross-Platform Toolchain (detta anche cross-toolchain) è una toolchain creata per essere utilizzata nell'architettura *host* e per produrre file binari eseguibili nell'architettura *target*. In questo modo è possibile sviluppare software eseguibile nell'architettura target sfruttando un ambiente di lavoro più performante costituito nella macchina host.

Dagli archivi della mailing list *arm-linux-toolchain* si apprende l'esistenza del progetto open-source *OSELAS Toolchain*[23] che fornisce un metodo di build integrato per una cross-toolchain adatta all'architettura target considerata.

[23] `http://www.pengutronix.de/oselas/toolchain/`

Questo progetto si avvale del sistema di build open source *Ptxdist* grazie al quale, oltre ad una cross-toolchain valida, è possibile compilare un'immagine del kernel personalizzata ed un root-filesystem valido per creare una piattaforma target completa.

Installare e configurare Ptxdist

Al momento della scrittura di questo documento la versione più recente è la 1.99.18[24]. Per scaricare ed installare ptxdist si procede in questo modo:

```
$ ~/bin/android.sh
$ mkdir $PRJROOT/build-tools
$ cd $PRJROOT/build-tools
$ wget http://www.pengutronix.de/software/ptxdist/download/v1
   .99/ptxdist-1.99.18.tgz
$ wget http://www.pengutronix.de/software/ptxdist/download/v1
   .99/ptxdist-1.99.18-patches.tgz
$ tar -xf ptxdist-1.99.18.tgz
$ tar -xf ptxdist-1.99.18-patches.tgz
$ cd ptxdist-1.99.18/
$ ./configure --prefix=$PRJROOT/build-tools
$ make
$ sudo make install
```

Perché il file binario ptxdist possa essere ricercato dalla shell il file *android.sh* verrà modificato in questo modo:

Listing 6.4: *android.sh* Ptxdist nel percorso di ricerca.

```
export PRJROOT=/media/dati/android
+export PATH=$PATH:$PRJROOT/build-tools/bin/

#Shell di lavoro
```

Il passo successivo è quello di configurare ptxdist:

```
$ ptxdist setup
```

Il menu risultante permette di configurare diversi aspetti del sistema di build:

Proxies : Se la macchina host risiede dietro Proxy di rete definirne le configurazioni.

[24] http://www.pengutronix.de/software/ptxdist/download/

Project Searchpath : Directory di ricerca per template di progetti (*$PRJROOT/tools*[25]).

Source Directory : Directory dove verranno scaricati tutti i pacchetti sorgente necessari al progetto (*$PRJROOT/build-tools/src*).

Source download : Per personalizzare la locazione dei repository da dove prelevare i sorgenti.

IPGK Repository : IPGK è un sistema di pacchettizzazione per applicazioni, come deb o rpm. Ptxdist così da l'opportunità di creare pacchetti per le applicazioni a supporto di una piattaforma già esistente.

Java SDK : Ptxdist permette di sviluppare pacchetti contenenti file binari Java per i quali è necessario impostare la locazione del SDK.

Salvate le configurazioni si procede nella costruzione della toolchain.

OSELAS toolchain

Il progetto per la cross-toolchain OSELAS può essere scaricato con:

```
$ cd $PRJROOT/build-tools
$ wget http://www.pengutronix.de/oselas/toolchain/download/
   OSELAS.Toolchain-1.99.3.4.tar.bz2
$ tar -xf OSELAS.Toolchain-1.99.3.4.tar.bz2 $ cd OSELAS.
   Toolchain-1.99.3.4/
```

Il pacchetto scaricato contiene diversi progetti ptxdist possibili per costruire toolchain, tutti risiedono nella sottodirectory ptxconfigs:

```
$ ls ptxconfigs/
arm-1136jfs-linux-gnueabigcc-4.1.2glibc-2.5binutils-2.17kernel
   -2.6.18.ptxconfig
arm-1136jfs-linux-gnueabigcc-4.3.2glibc-2.8binutils-2.19kernel
   -2.6.27-sanitized.ptxconfig
arm-cortexa8-linux-gnueabigcc-4.3.2glibc-2.8binutils-2.18kernel
   -2.6.27-sanitized.ptxconfig
armeb-xscale-linux-gnueabigcc-4.1.2glibc-2.5binutils-2.17kernel
   -2.6.18.ptxconfig
armeb-xscale-linux-gnueabigcc-4.3.2glibc-2.8binutils-2.18kernel
   -2.6.27-sanitized.ptxconfig
arm-hardfloat
arm-iwmmx-linux-gnueabigcc-4.1.2glibc-2.5binutils-2.17kernel
   -2.6.18.ptxconfig
```

[25] $PRJROOT deve essere espansa al suo valore.

```
arm-iwmmx-linux-gnueabigcc-4.3.2glibc-2.8binutils-2.18kernel
   -2.6.27-sanitized.ptxconfig
arm-oabi
arm-v4t-linux-gnueabigcc-4.1.2glibc-2.5binutils-2.17kernel
   -2.6.18.ptxconfig
arm-v4t-linux-gnueabigcc-4.3.2glibc-2.8binutils-2.18kernel
   -2.6.27-sanitized.ptxconfig
arm-v5te-linux-gnueabigcc-4.1.2glibc-2.5binutils-2.17kernel
   -2.6.18.ptxconfig
arm-v5te-linux-gnueabigcc-4.3.2glibc-2.8binutils-2.18kernel
   -2.6.27-sanitized.ptxconfig
arm-v5tevfp-linux-gnueabigcc-4.3.2glibc-2.8binutils-2.18kernel
   -2.6.27-sanitized.ptxconfig
arm-xscale-linux-gnueabigcc-4.1.2glibc-2.5binutils-2.17kernel
   -2.6.18.ptxconfig
arm-xscale-linux-gnueabigcc-4.2.3glibc-2.8binutils-2.18kernel
   -2.6.27-sanitized.ptxcon
fig
avr
i586-unknown-linux-gnugcc-4.1.2glibc-2.5binutils-2.17kernel
   -2.6.18.ptxconfig
i586-unknown-linux-gnugcc-4.3.2glibc-2.8binutils-2.18kernel
   -2.6.27-sanitized.ptxconfig
i686-unknown-linux-gnugcc-4.1.2glibc-2.5binutils-2.17kernel
   -2.6.18.ptxconfig
i686-unknown-linux-gnugcc-4.3.2glibc-2.8binutils-2.18kernel
   -2.6.27-sanitized.ptxconfig
java
mingw
mipsel-softfloat-linux-gnugcc-4.2.3glibc-2.8binutils-2.18kernel
   -2.6.27-sanitized.ptxconfig
newlib
```

Il progetto utile è

```
arm-1136jfs-linux-gnueabi_gcc-4.3.2_glibc-2.8_binutils-2.19
   _kernel-2.6.27-sanitized.ptxconfig
```

Selezionato e configurabile con le seguenti righe di codice:

```
$ ptxdist select ptxconfigs/arm-1136jfs-linux-gnueabigcc-4.3.2
   glibc-2.8 binutils-2.19kernel-2.6.27-sanitized.ptxconfig $
   ptxdist menuconfig
```

Le voci del menu:

Project Name : Personalizza il nome del progetto per costruire toolchain differenti (OSELAS.Toolchain-1.99.3).

architecture : Seleziona l'architettura target (arm).

toolchain target : Definisce il prefisso del nome per ogni utility secondo lo standard: cpu-manufacturer-kernel-os (arm-1136jfs-linux-gnueabi).

C library : Definisce quale libreria C utilizzare tra glibc, uClibc e altre (glibc).

glibc : Permette di configurare la compilazione delle librerie glibc.

glibc-ports : Permette di applicare una serie di patch a glibc relative ed un porting specifico.

binutils : Permette di configurare la compilazione delle utility binutils.

kernel : Permette di definire quale è la versione del kernel di riferimento (minima versione supportata).

gcc : Permette di configurare la compilazione di gcc utilizzato poi come cross-compilatore.

cross gdb : Se selezionata allora verrà creato il cross-debugger gdb.

misc : per definire alcuni parametri globali come directory di prefisso (`${PRJROOT}/tools`[26]) e versione di compatibilità di ptxdist.

Terminata la configurazione, si procede a dare il via al processo di build:

```
$ ptxdist go
```

Durante l'esecuzione è stata richiesta l'utility fakeroot installabile con:

```
$ sudo apt-get install fakeroot
```

Al termine del processo nella cartella:

${PRJROOT}/tools/OSELAS.Toolchain-1.99.3/arm-1136jfs-linux-gnueabi/gcc-4.3.2-glibc-2.8-binutils-2.19-kernel-2.6.27-sanitized/bin/ sono presenti i file binari della cross-toolchain creata.

6.2.5 Configurare e compilare il kernel

Gli script di configurazione del kernel sono sensibili a determinate variabili d'ambiente impostate nella shell corrente.

Per configurare un kernel per architettura ARM deve essere impostata nella la variabile d'ambiente `ARCH=arm`. Allora con:

[26] ${PRJROOT} deve essere espansa al suo valore.

```
$ cd $PRJROOT/kernelLinus/linux-2.6/
$ ARCH=arm
$ make "familydefconfig"
$ make menuconfig -> Text Based
or $ make gconfig -> Gtk Based
or $ make xconfig -> Qt Based
```

Potranno essere scelte tutte le caratteristiche volute del kernel sulla base del file di configurazione di default specifico per la macchina utilizzata. Nel caso della famiglia di Application Processor i.MX il file di configurazione standard è: `make mx3_defconfig`.

Le funzionalità del kernel vengono selezionate attraverso voci di menu a due o tre stati dove il simbolo `*` indica la presenza statica nell'immagine finale mentre il simbolo `M` indica la creazione di un modulo esterno che il kernel potrà caricare durante l'esecuzione.

Configurato il kernel, il processo di build deve essere istruito per utilizzare la cross-toolchain corretta creata precedentemente. Per questo deve essere impostata la variabile d'ambiente `CRPSS_COMPILE=toolchain-prefix` con `toolchain-prefix` prefisso completo dei binari della toolchain.

Dato che queste due variabili sono globali a tutto il progetto vengono inserite in *`android.sh`* con le righe:

Listing 6.5: *`android.sh`* Configurazione dell'ambiente di compilazione ARM.

```
export PATH=$PATH:$PRJROOT/build-tools/bin/

+#Per compilare kernel ARM
+export ARCH=arm
+export CROSSCOMPILE=$PRJROOT/tools/OSELAS.Toolchain-1.99.3/\
+arm-1136jfs-linux-gnueabi/gcc-4.3.2-glibc-2.8-binutils-2.19-kernel-2.6.27-
    sanitized/bin/arm-1136jfs-linux-gnueabi-

#Shell di lavoro
```

Una volta impostate correttamente le variabili d'ambiente, il kernel può essere compilato:

```
$ make
```

Al termine del processo di build in *`$PRJROOT/kernelLinus/linux-2.6/arch/arm/boot/`* sarà presente il file zImage, immagine compressa del kernel, da scaricare nella macchina target mentre i moduli creati dovranno essere installati nella root directory scelta (*`$PRJROOT/rootfs`*) con:

```
$ make modules_install INSTALLMODPATH=$PRJROOT/rootfs
```

6.2.6 Creare un root filesystem valido

Il progetto OSELAS[27] gestito da Pengutronix è di più ampio respiro: oltre a fornire un metodo automatizzato per creare una toolchain adatta per molte architetture ARM esistono progetti ptxdist per creare root filesystem completi e se supportate, fornisce utility per la scrittura nelle flash della board target.

Ciò che interessa per questo lavoro di tesi è la creazione di un root filesystem valido utilizzabile successivamente per le operazioni di test e debug.

A questo proposito è disponibile il progetto *OSELAS BSP* mantenuto da Phytec.

Da una nuova shell:

```
$ ~/bin/android.sh
$ mkdir $PRJROOT/build-BSP
$ cd $PRJROOT/build-BSP
$ wget http://www.oselas.com/oselas/bsp/phytec/download/phyCORE/
   OSELAS.BSP- Phytec-phyCORE-12-1.tar.gz
$ tar -xf OSELAS.BSP-Phytec-phyCORE-12-1.tar.gz
$ cd OSELAS.BSP-Phytec-phyCORE-12-1
```

Con le seguenti ptxdist viene correttamente configurato con il file di progetto ed un particolare file di piattaforma:

```
$ ptxdist select configs/ptxconfig
$ ptxdist platform configs/phyCORE-i.MX31-1.99.12-3/
   platformconfig
```

Dato che la directory di installazione della toolchain non è standard, deve essere impostata la corretta posizione:

```
$ ptxdist toolchain
$PRJROOT/tools/OSELAS.Toolchain-1.99.3/arm-1136jfs-linux-gnueabi
   /gcc-4.3.2-glibc-2.8-binutils-2.19-kernel-2.6.27-sanitized/
   bin/
```

Con il termine *piattaforma* ptxdist considera l'insieme kernel, librerie C e un root filesystem di base che fornisca la struttura contenitore alle applicazioni.

[27] Open Source Embedded Linux Automation Services `http://www.oselas.com`

Con la seguente riga di comando possono essere personalizzate le configurazioni di piattaforma (`-force` è inserito per compatibilità tra le versioni differenti di ptxdist):

```
$ ptxdist --force platformconfig
```

Nel menu risultante deve essere deselezionato `Linux Kernel` dato che un kernel apposito verrà sviluppato nel proseguo del lavoro. Importante poi è il sotto menu "`image creation option`" dove è definito il formato di output dell'immagine di root, che può essere nelle forme:

hd.img : Contiene il boot loader grub, il kernel, ed il root filesystem in una partizione ext2.

root.jffs2 : Directory root in una partizione jffs2.

uRamdisk : Directory root in una immagine Ramdisk compatibile con u-boot.

initrd.gz : Tradizionale immagine RAM initrd utilizzabile dal kernel come initrd ramfs.

root.ext2 : Directory root in una partizione ext2.

root.squashfs : Directory root in una partizione squashfs.

root.tgz : Directory root in un file compresso gzip (unica da selezionare).

Configurata la piattaforma si procede alla configurazione della Board Support Package considerata come l'insieme di tutto il software che verrà installato nella root directory:

```
$ ptxdist --force menuconfig
```

In `Project Specific Configuration` deselezionare: `phyCORE-MPC5200B simple FPGA loader`

In `PTXdist Base Configuration` è possibile configurare la struttura del root filesystem e le applicazioni che vi verranno installate.

Le applicazioni vengono selezionate attraverso voci di menu a tre stati dove il simbolo `*` indica la presenza di default nell'immagine mentre il simbolo `M` indica la creazione di un pacchetto IPGK scaricabile ed installabile nella macchina target.

Mentre tutto il resto delle opzioni può rimanere come già impostato (verrà creato un root filesystem molto leggero basato su BusyBox[28]), saranno

[28] `http://www.busybox.net/`

utili per questo lavoro di tesi le utility `Graphics & Multimedia -> framebuffer -> fbtest`, `fbutils`, `fbv` da includere tutte e tre con il simbolo `*` (default).

Ora è possibile partire con il processo di build:

```
$ ptxdist go
```

Ptxdist, utilizzando le informazioni in *selected_ptxconfig* e *selected_platformconfig* (link simbolici ai file di configurazione selezionati), automaticamente provvederà a scaricare, compilare ed installare i pacchetti necessari tenendo conto delle dipendenze specifiche per ogni pacchetto.

Al termine del processo di build il root filesystem si troverà nella directory *platform-phyCORE-i.MX31/root/* mentre in *platform-phyCORE-i.MX31/packages* risiederanno tutti i file pacchetto nel formato IPGK (.ipk) delle applicazioni selezionate con il simbolo `M`.

La seguente riga di comando creerà l'immagine del root filesystem nella directory *platform-phyCORE-i.MX31/images* :

```
$ ptxdist images
```

e con le seguenti, l'immagine sarà disponibile nella directory *$PRJROOT/rootfs*:

```
$ mkdir $PRJROOT/rootfs
$ cp platform-phyCORE-i.MX31/images/root.tgz $PRJROOT/rootfs
$ cd $PRJROOT/rootfs
$ sudo tar -xf root.tgz
$ sudo rm root.tgz
```

Importante è il file di configurazione della rete della macchina target *$PRJROOT/rootfs/etc/network/interfaces* da modificare a seconda delle impostazioni della rete utilizzata per lo sviluppo. Ad esempio:

Listing 6.6: Template per il file *$PRJROOT/rootfs/etc/network/interfaces*

```
#
# /etc/network/interfaces
#

auto lo eth0

# loopback interface
iface lo inet loopback

# ethernet iface eth0
inet static
        address 192.168.0.10
        netmask 255.255.255.0
```

6.2.7 La connessione con la macchina target

minicom

Con la seguente riga di comando viene installato nella macchina host il programma *minicom*[29]:

```
$ sudo apt-get install minicom
```

minicom è un terminale per comunicazioni seriali sviluppato per sistemi POSIX grazie al quale sarà possibile interagire con la macchina target.

Per configurare le impostazioni del programma:

```
$ minicom -s
```

Dove è necessario configurare i parametri della connessione seriale in `Serial port setup` sulla base delle impostazioni della board riportate nella sezione 5.2.1:

Opzione	**Valore**	**Note**
Serial Device :	/dev/ttyUSB0	Per cavi USB/Seriali altrimenti /dev/ttyN
Lockfile Location :	/var/lock	
Callin Program :	No	
Callout Program :	No	
Bps/Par/Bits :	115200 8N1	115200bps 8bit lenght No parity 1bit stop
Hardware Flow Control :	No	
Software Flow Control :	No	

tftp server

Per scaricare file nella flash della board tramite boot loader è necessario installare un server tftp nella macchina host:

```
$ sudo apt-get install tftpd-hpa
```

Con le seguenti viene creata una cartella di scambio e dato l'accesso in lettura e scrittura ai normali utenti:

```
$ sudo mkdir /tftp
```

[29] `http://alioth.debian.org/projects/minicom`

```
$ sudo chmod a+r /tftp
$ sudo chmod a+w /tftp
```

Proseguendo nella configurazione del demone, deve essere modificato il file di configurazione dei servizi */etc/inet.conf* aggiungendo la riga:

```
#servicename sockettype protocol wait/nowait user serverprogram serverprogarguments
...
+ tftp dgram udp wait root /usr/sbin/in.tftpd /usr/sbin/in.tftpd -s /tftp
```

L'argomento -s indica al demone tftpd di utilizzare la directory */tftp* come root directory, se si vuole ottenere maggiori informazioni di debug aggiungere -v.

Se non avviati nella fase di boot del sistema, con:

```
$ sudo inetd
```

Verranno attivati i servizi configurati in */etc/inet.conf* assieme al server tftp. Specificando l'opzione `-d` verranno visualizzate le stringhe di debug dei servizi nella shell chiamante.

nfs server

Nelle operazioni successive di test e debug sarà utile installare un server nfs nella macchina host per poter utilizzare come root filesystem della macchina target una directory remota:

```
$ sudo apt-get install nfs-kernel-server
```

Installerà oltre ai binari, il proprio script di inizializzazione in */etc/init.d/* per l'avvio automatico durante il boot della macchina host.

Il server è configurato attraverso il file */etc/exports* dove ogni riga è nella forma:

```
# homepath hostname1(options) hostname2(options)
```

home_path : è il percorso assoluto della cartella che un client remoto può montare.

hostnameN : è un nome di host valido al quale è concesso l'accesso. Questo può essere espresso anche come singolo ip o indirizzo di sotto rete.

options : le opzioni di mount utilizzate per vincolare l'accesso (ro,rw), disabilitare la cache in scrittura (sync), modificare la traduzione uid/gid (no_root_squash) etc... per una lista completa delle opzioni si rimanda alla pagina man di exports.

Opzioni corrette per un root filesystem sono: (rw,no_all_squash,no_root_squash).

Allora, per rendere disponibile la cartella */$PRJROOT/rootfs* dovrà errere scritta la riga:

```
/$PRJROOT/rootfs 192.168.0.1/255.255.255.0(rw,noallsquash,norootsquash)
```

Con `$PRJROOT` Espansa al path reale (*/media/dati/android*).

L'indirizzo della sotto rete dei client permessi deve essere compatibile con le impostazioni della sotto rete di connessione tra macchina host e board. L'home_path qui definito dovrà essere riportato nei parametri del kernel nel momento in cui si vorrà eseguire il boot via rete.

6.2.8 Scaricare il kernel nella board Armadillo 500

Nel paragrafo 6.2.7 è stato installato nella macchina host il server tftp necessario per scaricare file nella memoria flash del modulo CPU della board considerata ed alla fine del paragrafo 6.2.5 si è mostrato come compilare un nuovo kernel.

Ora verranno descritti i passi per scaricare il nuovo kernel compilato nella regione di boot della board:

Nella shell di compilazione del kernel (successivamente l'esecuzione di `make`):

```
$ cp arch/arm/boot/zImage /tftp/linux.bin.gz
```

Collegata la board alla macchina host attraverso il cavo seriale e tramite la rete ethernet, in una nuova shell:

```
$ minicom ...

hermit> tftpdl 192.168.0.10 192.168.0.2 --kernel=linux.bin.gz
```

Con `192.168.0.10` indirizzo ip assegnato alla board e `192.168.0.2` indirizzo ip dell'interfaccia di rete della workstation.

Al termine del trasferimento la riga seguente darà il comando di procedere nella sequenza di boot della board con i parametri del kernel precedentemente definiti:

```
hermit> boot
```

6.3 Fasi del porting

Qui di seguito verranno presentate tutte le fasi che hanno portato alla definizione del codice di piattaforma sviluppato.

Questa sequenza è stata applicata al repository locale del branch *mxc-master* del kernel Linux[30] a partire dalla versione 2.6.29.

Ogni fase ha prodotto patch specifiche che, dopo il processo di review del gruppo MXC della comunità ARM-Linux, sono state inserite nel repository Pengutronix ed hanno proseguito la strada verso il mainline kernel.

Le piattaforme simili presenti nella sottodirectory del kernel tree */arch/arm/mach-mx3* ed i sorgenti del branch Atmark del kernel[31] sono state utili linee guida per il lavoro svolto e qui documentato.

6.3.1 Early support

Come mostrato nella sequenza di boot su architettura ARM nella sezione 6.1.3, il kernel utilizza il parametro id di macchina `MACH_TYPE` per ricercare una struttura dati di tipo `machine_desc` che contenga il codice di inizializzazione della piattaforma hardware presente.

La struttura dati `machine_desc` è definita nel file */arch/arm/include/asm/mach/arch.h*:

Listing 6.7: */arch/arm/include/asm/mach/arch.h*

```
/*
 *  arch/arm/include/asm/mach/arch.h
 *
 *  Copyright (C) 2000 Russell King
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#ifndef __ASSEMBLY__
```

[30] Si faccia riferimento alla sezione 6.2.2 per l'inizializzazione di questo repository.

[31] `http://download.atmark-techno.com/armadillo-500/`

```
struct tag;
struct meminfo;
struct sys_timer;

struct machine_desc {
        /*
         * Note! The first four elements are used
         * by assembler code in head.S, head-common.S
         */
        unsigned int            nr;             /* architecture number  */
        unsigned int            phys_io;        /* start of physical io */
        unsigned int            io_pg_offst;    /* byte offset for io
                                                 * page tabe entry      */

        const char              *name;          /* architecture name    */
        unsigned long           boot_params;    /* tagged list          */

        unsigned int            video_start;    /* start of video RAM   */
        unsigned int            video_end;      /* end of video RAM     */

        unsigned int            reserve_lp0 :1; /* never has lp0        */
        unsigned int            reserve_lp1 :1; /* never has lp1        */
        unsigned int            reserve_lp2 :1; /* never has lp2        */
        unsigned int            soft_reboot :1; /* soft reboot          */
        void                    (*fixup)(struct machine_desc *,
                                         struct tag *, char **,
                                         struct meminfo *);
        void                    (*map_io)(void);/* IO mapping function  */
        void                    (*init_irq)(void);
        struct sys_timer        *timer;         /* system tick timer    */
        void                    (*init_machine)(void);
};

/*
 * Set of macros to define architecture features.  This is built into
 * a table by the linker.
 */
#define MACHINE_START(_type,_name)                      \
static const struct machine_desc __mach_desc_##_type   \
 __used                                                 \
 __attribute__((__section__(".arch.info.init"))) = {    \
        .nr             = MACH_TYPE_##_type,            \
        .name           = _name,

#define MACHINE_END                             \
};

#endif
```

Mentre gli id di macchina sono registrati nel file */arch/arm/tools/mach-types*.

Lo scheletro della piattaforma di supporto quindi è composto da:

Un ID di macchina appropriato. Atmark ha già registrato in passato l'id `ARMADILLO5X0` per la board Armadillo 500 ed ha codificato lo stesso valore nel boot loader Hermit At:

Listing 6.8: Estratto del file */arch/arm/tools/mach-types*.

```
# machine_is_xxx     CONFIG_xxxx         MACH_TYPE_xxx         number
  ...
armadillo5x0         MACH_ARMADILLO5X0    ARMADILLO5X0          1260
```

Per questo la piattaforma di supporto qui creata utilizzerà lo stesso identificatore `MACH_TYPE_ARMADILLO5X0` senza la necessità di registrarne uno nuovo.

Il sorgente di supporto. All'interno della directory */arch/arm/mach-mx3* deve essere creato il file *armadillo5x0.c* contenente una implementazione di base della struttura dati `machine_desc`:

Listing 6.9: */arch/arm/mach-mx3/armadillo5x0.c*

```
/*
 * armadillo5x0.c
 *
 * Copyright 2009 Alberto Panizzo <maramaopercheseimorto@gmail.com>
 * updates in http://alberdroid.blogspot.com/
 *
 * Based on Atmark Techno, Inc. armadillo 500 BSP 2008
 * Based on mx31ads.c and pcm037.c Great Work!
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 */

#include <linux/types.h>
#include <linux/init.h>
#include <linux/clk.h>

#include <mach/hardware.h>
#include <asm/mach-types.h>
#include <asm/mach/arch.h>
#include <asm/mach/time.h>
#include <asm/memory.h>
#include <asm/mach/map.h>

#include <mach/common.h>
#include <mach/board-armadillo5x0.h>
```

```
/*
 * Perform board specific initializations
 */
static void __init armadillo5x0_init(void)
{
}

static void __init armadillo5x0_timer_init(void)
{
        mx31_clocks_init(26000000);
}

static struct sys_timer armadillo5x0_timer = {
        .init   = armadillo5x0_timer_init,
};

MACHINE_START(ARMADILLO5X0, "Armadillo-500")
        /* Maintainer: Alberto Panizzo  */
        .phys_io        = AIPS1_BASE_ADDR,
        .io_pg_offst    = ((AIPS1_BASE_ADDR_VIRT) >> 18) & 0xfffc,
        .boot_params    = PHYS_OFFSET + 0x00000100,
        .map_io         = mx31_map_io,
        .init_irq       = mx31_init_irq,
        .timer          = &armadillo5x0_timer,
        .init_machine   = armadillo5x0_init,
MACHINE_END
```

I parametri da `.phys_io` a `.init_irq` sono definiti con valori di piattaforma standard per l'Application Processor i.MX31:

`mx31_map_io` mappa staticamente le aree di indirizzi dell'AVIC, della memoria centrale e degli AIPS1 e 2.

`mx31_init_irq` inizializza i registri di controllo dell'AVIC.

Il parametro `.timer` punta ad una struttura `sys_timer` contenente le informazioni necessarie ad inizializzare il sottosistema di sorgenti clock di Linux:

`int __init mx31_clocks_init(unsigned long fref)` imposta il valore della frequenza di riferimento in ingresso alla CPU (`ckih_rate` frequenza di generazione di tutte le sorgenti di clock dell'Application Processor) ed inizializza l'interfaccia `clk_dev` del kernel Linux.

Il parametro `.init_machine` punta alla funzione che inizializzerà in seguito tutti i device della board che per ora rimane vuota.

Il codice di collegamento al processo di build del kernel. I file *Kconfig* e *Makefile* interni alla directory */arch/arm/mach-mx3* devono essere modificati per permettere di scegliere e compilare la nuova piattaforma:

Listing 6.10: Patch ai file Kconfig e Makefile in */arch/arm/mach-mx3*

```
diff --git a/arch/arm/mach-mx3/Kconfig b/arch/arm/mach-mx3/Kconfig
index d623558..21b712a 100644
--- a/arch/arm/mach-mx3/Kconfig
+++ b/arch/arm/mach-mx3/Kconfig
@@ -64,4 +64,11 @@ config MACH_QONG
          Include support for Dave/DENX QongEVB-LITE platform. This includes
          specific configurations for the board and its peripherals.

+config MACH_ARMADILLO5X0
+       bool "Support Atmark Armadillo-500 Development Base Board"
+       select ARCH_MX31
+       help
+         Include support for Atmark Armadillo-500 platform. This includes
+         specific configurations for the board and its peripherals.
+
 endif
diff --git a/arch/arm/mach-mx3/Makefile b/arch/arm/mach-mx3/Makefile
index 272c8a9..b93bfa7 100644
--- a/arch/arm/mach-mx3/Makefile
+++ b/arch/arm/mach-mx3/Makefile
@@ -14,3 +14,4 @@ obj-$(CONFIG_MACH_MX31_3DS)    += mx31pdk.o
 obj-$(CONFIG_MACH_MX31MOBOARD) += mx31moboard.o mx31moboard-devboard.o \
                                   mx31moboard-marxbot.o
 obj-$(CONFIG_MACH_QONG)        += qong.o
+obj-$(CONFIG_MACH_ARMADILLO5X0)        += armadillo5x0.o
```

Early debug. Per permettere il debug nella prima fase di boot del kernel (si veda la sezione 6.1.4.1), oltre allo scheletro di base devono essere inserite le seguenti modifiche nel codice di piattaforma della famiglia MXC:

In */arch/arm/plat-mxc/include/mach/* deve essere creato un file header apposito chiamato *board-armadillo5x0.h* contenente le definizioni riguardo l'indirizzo del dispositivo seriale da utilizzare nelle operazioni di early debug.

Listing 6.11: */arch/arm/plat-mxc/include/mach/board-armadillo5x0.h*

```
/*
 * Copyright 2009 Alberto Panizzo <maramaopercheseimorto@gmail.com>.
 * All Rights Reserved.
 */
```

```
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#ifndef __ASM_ARCH_MXC_BOARD_ARMADILLO5X0_H__
#define __ASM_ARCH_MXC_BOARD_ARMADILLO5X0_H__

#include <mach/hardware.h>

/* mandatory for CONFIG_DEBUG_LL */

#define MXC_LL_UART_PADDR       UART1_BASE_ADDR
#define MXC_LL_UART_VADDR       AIPS1_IO_ADDRESS(UART1_BASE_ADDR)

#endif
```

Nel file */arch/arm/plat-mxc/include/mach/debug-macro.S* deve essere inserito il collegamento al file header creato perché le routine **printascii** possa utilizzare il device seriale nel caso venga selezionata la macchina `ARMADILLO5x0`.

Listing 6.12: Patch al file */arch/arm/plat-mxc/include/mach/debug-macro.S*

```
diff --git a/arch/arm/plat-mxc/include/mach/debug-macro.S
b/arch/arm/plat-mxc/include/mach/debug-macro.S
index 4f77314..af72235 100644
--- a/arch/arm/plat-mxc/include/mach/debug-macro.S
+++ b/arch/arm/plat-mxc/include/mach/debug-macro.S
@@ -34,6 +34,9 @@
 #ifdef CONFIG_MACH_QONG
 #include <mach/board-qong.h>
 #endif
+#ifdef CONFIG_MACH_ARMADILLO5X0
+#include <mach/board-armadillo5x0.h>
+#endif
                .macro  addruart,rx
                mrc     p15, 0, \rx, c1, c0
                tst     \rx, #1                 @ MMU enabled?
```

Creazione dell'immagine. Così modificati i sorgenti del kernel ed applicata la patch per il low level debug (listato 6.1 con `git apply nome_patch`) si procede alla configurazione e compilazione come descritto nel paragrafo 6.2.5 basando la configurazione sul file di default `mx3_defconfig` e selezionando l'architettura appena creata nel sotto-menu di configurazione del kernel:

`System Type -->` selezioneare `ARM system type (Freescale MXC/iMX-based)` e proseguendo in `Freescale MXC Implementations`

`-->` selezionare `Support Atmark Armadillo-500 Development Base Board`.

Per abilitare il debug low level: `Kernel hacking -->` selezionare `Kernel low-level debugging functions`.

Scaricata l'immagine del kernel così ottenuta nella board come descritto nel paragrafo 6.2.8 e dato il comando di boot, si potranno vedere nel terminale minicom le prime righe di debug del kernel riguardo l'inizializzazione della CPU ed operazioni iniziali fino all'errore fatale riguardo la mancanza di una console di sistema.

6.3.2 Console attraverso la porta seriale

Il kernel necessita della console di sistema innanzitutto per presentare l'output di debug e successivamente per poter instaurare una possibile interfaccia interattiva di controllo per l'utente.

Per la piattaforma MXC è stato sviluppato da Sascha Hauer <*sascha@saschahauer.de*> il driver *drivers/serial/imx.c* a supporto dei dispositivi UART della famiglia di Application Processor i.MX che, oltre ad abilitare il canale di comunicazione seriale, implementa le API di console di sistema permettendo così la comunicazione di debug e controllo da e per la macchina target.

Per far si che il kernel riconosca e configuri correttamente i due dispositivi seriali `UART1` e `UART2` (vedi la sezione 5.1.2) è necessario:

Configurare correttamente le funzioni dei pin di comunicazione associati ai due device UART. Per questo esiste la funzione di piattaforma:

`int mxc_iomux_setup_multiple_pins(unsigned int *pin_list, unsigned count, const char *label)`

Definita nel file *arch/arm/plat-mxc/include/mach/iomux.c* che dato in ingresso un array di identificatori di pin `pin_list` ne registra l'assegnazione in una tabella di allocazione propria di piattaforma (`mxc_pin_alloc_map`) e ne imposta uno ad uno la funzione richiesta intervenendo nei registri di configurazione del IOMUX Controller interno all'Application Processor[33, Cap. 4] con il metodo **`mxc_iomux_mode()`**.

Gli id che identificano ogni pin del processore sono definiti nel file header

arch/arm/plat-mxc/include/mach/iomux-mx3.h assieme alle macro di modificazione della funzione.

Registrare i due dispositivi nel driver model Linux. A questo proposito esiste la funzione:

int __init mxc_register_device(struct platform_device *pdev, void *data)

Wrapper della funzione standard:

int platform_device_register(struct platform_device *pdev)

Dove il dispositivo di piattaforma è descritto da una struttura `platform_device` definita come:

```
struct platform_device {
        const char      * name;
        int             id;
        struct device   dev;
        u32             num_resources;
        struct resource * resource;

        struct platform_device_id       *id_entry;

        /* arch specific additions */
        struct pdev_archdata    archdata;
};
```

I membri di questa struttura rilevanti nella registrazione di un dispositivo di piattaforma sono:

- Il nome di dispositivo permette di ricercare il driver corretto tra quelli registrati.
- L'`id` specifica, nel caso siano disponibili più dispositivi dello stesso tipo, quale deve essere considerato. -1 se la scelta è univoca.
- Le risorse elencate in un array di dimensione `num_resources` che possono indicare al driver: l'area di memoria dove sono mappati i registri di controllo e configurazione (tag `IORESOURCE_MEM`), gli identificativi di interrupt assegnati (tag `IORESOURCE_IRQ`) ed i buffer di memoria DMA associati (tag `IORESOURCE_DMA`) [32].
- `archdata` infine permette di passare al driver dei parametri specifici riguardo le modalità di interfacciamento hardware utilizzata.

[32] Il file */include/linux/ioport.h* contiene le definizioni di tutti i possibili tag di risorsa espressi nel parametro `flag` della struttura `resource`.

Una descrizione più approfondita della gestione dei dispositivi di piattaforma può essere trovata nel documento [9] parte della definizione del Linux Driver Model [14].

Il codice di piattaforma definisce nel file */arch/arm/mach-mx3/devices.c* una specifica struttura `platform_device` per ogni device interno agli Application Processor i.MX31 completa delle risorse standard.

Allora per i due dispositivi `UART1` e `UART2` devono essere registrati i descrittori `mxc_uart_device0` e `mxc_uart_device1`.

Modifiche al codice di supporto. Di seguito sono riportate le modifiche al file *armadillo5x0.c*:

Listing 6.13: armadillo5x0-01ToUART.patch

```
***************
*** 17,22 ****
--- 17,23 ----
  #include <linux/types.h>
  #include <linux/init.h>
  #include <linux/clk.h>
+ #include <linux/platform_device.h>

  #include <mach/hardware.h>
  #include <asm/mach-types.h>
***************
*** 26,38 ****
--- 27,67 ----
  #include <asm/mach/map.h>

  #include <mach/common.h>
+ #include <mach/imx-uart.h>
+ #include <mach/iomux-mx3.h>
  #include <mach/board-armadillo5x0.h>

+ #include "devices.h"
+
+ static int armadillo5x0_pins[] = {
+         /* UART1 */
+         MX31_PIN_CTS1__CTS1,
+         MX31_PIN_RTS1__RTS1,
+         MX31_PIN_TXD1__TXD1,
+         MX31_PIN_RXD1__RXD1,
+         /* UART2 */
+         MX31_PIN_CTS2__CTS2,
+         MX31_PIN_RTS2__RTS2,
+         MX31_PIN_TXD2__TXD2,
+         MX31_PIN_RXD2__RXD2,
+ };
```

```
+
+ /* UART device data */
+ static struct imxuart_platform_data uart_pdata = {
+       .flags = IMXUART_HAVE_RTSCTS,
+ };
+
  /*
   * Perform board specific initializations
   */
  static void __init armadillo5x0_init(void)
  {
+       mxc_iomux_setup_multiple_pins(armadillo5x0_pins,
+                       ARRAY_SIZE(armadillo5x0_pins), "armadillo5x0");
+
+       /* Register UART */
+       mxc_register_device(&mxc_uart_device0, &uart_pdata);
+       mxc_register_device(&mxc_uart_device1, &uart_pdata);
  }

  static void __init armadillo5x0_timer_init(void)
```

Abilitare la compilazione del driver seriale *imx.c* con supporto alla console. Nel menu di configurazione del kernel devono essere selezionate per la presenza statica le due opzioni: `Device Drivers --> Character devices --> Serial drivers --> IMX serial port support` e `Console on IMX serial port`.

Mentre deve essere disabilitata l'opzione di debug low level `Kernel hacking --> Kernel low-level debugging functions` per evitare conflitti di scrittura nel canale seriale.

Compilato e scaricato il kernel così modificato (come descritto nella fase precedente), nel menu di Hermit deve essere impostato il parametro del kernel perché il nuovo driver selezionato venga utilizzato come console di sistema:

```
hermit> setenv console=ttymxc0
```

Procedendo con l'operazione di boot della board si potranno vedere le stringhe di debug nella nuova console seriale fino all'errore fatale riguardo la mancanza di un root filesystem valido.

6.3.3 Rete Ethernet

La board Armadillo 500 monta un controller ethernet SMSC Lan 9118 (vedi la sezione 5.1.3), controller diffuso nei sistemi embedded e supportato dal driver in kernel */drivers/net/smsc911x.c*.

Gli schemi logici della main-board [22, foglio 3], i datasheet del dispositivo [18] e la memory map in appendice A forniscono la conoscenza necessaria per definirne le risorse ed i parametri di configurazione corretti del device.

Risorse del dispositivo. Devono contenere la zona di memoria dove è stato mappato lo spazio di indirizzamento dei registri interni del controller e l'identificativo di interrupt al quale è stato connesso il segnale `LAN_IRQ`:

Il controller SMSC 9118 è mappato nella terza regione di espansione degli indirizzi *CS3* (il pin di Chip select è connesso al segnale `CS3`) con uno spazio di indirizzamento di 32 MB.

La sorgente di interrupt `LAN_IRQ` è stata connessa al pin `GPIO1_0` dell'Application Processor i.MX31.

Allora le risorse sono così definite:

```
static struct resource armadillo5x0_smc911x_resources[] = {
        {
                .start  = CS3_BASE_ADDR,
                .end    = CS3_BASE_ADDR + SZ_32M - 1,
                .flags  = IORESOURCE_MEM,
        }, {
                .start  = IOMUX_TO_IRQ(MX31_PIN_GPIO1_0),
                .end    = IOMUX_TO_IRQ(MX31_PIN_GPIO1_0),
                .flags  = IORESOURCE_IRQ | IORESOURCE_IRQ_LOWLEVEL,
        },
};
```

La macro `IOMUX_TO_IRQ()` trasforma l'identificatore di pin del processore `MX31_PIN_GPIO1_0` in un identificatore di interrupt. Il modificatore di flag di risorsa `IORESOURCE_IRQ_LOWLEVEL` indica al driver lo stato attivo del segnale di interrupt che si sceglie essere quello basso.

Parametri del driver. Nel file */include/linux/smsc911x.h* sono definiti i parametri possibili di configurazione del driver smsc9118x che permettono le impostazioni seguenti del controller:

- Larghezza del bus di dati: 16 o 32 bit.
- Quale modulo di gestione del livello fisico utilizzare (*phy interface*): interno oppure esterno (forzandolo).
- Semantica del segnale di interrupt: attivo alto oppure basso.

- Configurazione elettronica del circuito di interrupt: Push/Pull oppure Open Drain.

Il controller è collegato al bus dati di sistema attraverso i soli 16 bit LSB ed i pin di interfaccia verso la rete ethernet sono connessi direttamente al connettore RJ-45 (viene utilizzato il modulo di gestione del layer fisico interno). Allora il flag di configurazione del driver è determinato dal solo modificatore:`SMSC911X_USE_16BIT`.

Il segnale di interrupt è collegato al processore attraverso un circuito con resistenza di Pull Up. Questo per rendere possibili tutte e due le configurazioni circuitali interne al controller: Push/Pull e Open Drain[33] nel caso la semantica dell'interrupt lo vuole attivo basso.

Test effettuati hanno mostrato come la configurazione Open Drain risulti comunque in errori nella generazione del segnale per la semantica scelta, per cui la configurazione preferita è Push/Pull.

Allora i parametri di configurazione sono definiti in questo modo:

```
static struct smsc911x_platform_config smsc911x_info = {
        .flags          = SMSC911X_USE_16BIT,
        .irq_polarity   = SMSC911X_IRQ_POLARITY_ACTIVE_LOW,
        .irq_type       = SMSC911X_IRQ_TYPE_PUSH_PULL,
};
```

Configurazione della sorgente di interrupt. Il segnale di interrupt `LAN_IRQ` è connesso al pin gpio `GPIO1_0` dell'Application Processor i.MX31. Allora tale gpio deve essere configurato come input ed essere collegato al circuito interno di generazione degli interrupt.

Per ottenere questo risultato, deve essere registrato l'identificatore di pin `MX31_PIN_GPIO1_0` modificato nella funzione `IOMUX_CONFIG_GPIO` ed impostato nella direzione con **`gpio_direction_input()`**.

Il documento [7] fornisce una descrizione completa riguardante la gestione dei pin gpio nel kernel Linux.

Modifiche al codice di supporto. Il file *`armadillo5x0.c`* allora viene così modificato:

[33]Nel caso Push/Pull il controller è in grado di generare entrambi gli stati, alto e basso (1 o 0), mentre nel caso Open Drain è la resistenza esterna di pull up che provvede ad alzare il livello del segnale quando il circuito interno viene spento.

Listing 6.14: armadillo5x0-02ToEthernet.patch

```
***************
*** 18,23 ****
--- 18,27 ----
  #include <linux/init.h>
  #include <linux/clk.h>
  #include <linux/platform_device.h>
+ #include <linux/gpio.h>
+ #include <linux/smsc911x.h>
+ #include <linux/interrupt.h>
+ #include <linux/irq.h>

  #include <mach/hardware.h>
  #include <asm/mach-types.h>
***************
*** 44,49 ****
--- 48,87 ----
        MX31_PIN_RTS2__RTS2,
        MX31_PIN_TXD2__TXD2,
        MX31_PIN_RXD2__RXD2,
+       /* LAN9118_IRQ */
+       IOMUX_MODE(MX31_PIN_GPIO1_0, IOMUX_CONFIG_GPIO),
+ };
+
+ /*
+  * SMSC 9118
+  * Network support
+  */
+ static struct resource armadillo5x0_smc911x_resources[] = {
+       {
+               .start  = CS3_BASE_ADDR,
+               .end    = CS3_BASE_ADDR + SZ_32M - 1,
+               .flags  = IORESOURCE_MEM,
+       }, {
+               .start  = IOMUX_TO_IRQ(MX31_PIN_GPIO1_0),
+               .end    = IOMUX_TO_IRQ(MX31_PIN_GPIO1_0),
+               .flags  = IORESOURCE_IRQ | IORESOURCE_IRQ_LOWLEVEL,
+       },
+ };
+
+ static struct smsc911x_platform_config smsc911x_info = {
+       .flags          = SMSC911X_USE_16BIT,
+       .irq_polarity   = SMSC911X_IRQ_POLARITY_ACTIVE_LOW,
+       .irq_type       = SMSC911X_IRQ_TYPE_PUSH_PULL,
+ };
+
+ static struct platform_device armadillo5x0_smc911x_device = {
+       .name           = "smsc911x",
+       .id             = -1,
+       .num_resources  = ARRAY_SIZE(armadillo5x0_smc911x_resources),
+       .resource       = armadillo5x0_smc911x_resources,
+       .dev            = {
+               .platform_data = &smsc911x_info,
+       },
  };
```

```
  /* UART device data */
***************
*** 51,56 ****
--- 89,98 ----
        .flags = IMXUART_HAVE_RTSCTS,
  };

+ static struct platform_device *devices[] __initdata = {
+       &armadillo5x0_smc911x_device,
+ };
+
  /*
   * Perform board specific initializations
   */
***************
*** 59,67 ****
--- 101,114 ----
        mxc_iomux_setup_multiple_pins(armadillo5x0_pins,
                        ARRAY_SIZE(armadillo5x0_pins), "armadillo5x0");

+       platform_add_devices(devices, ARRAY_SIZE(devices));
+
        /* Register UART */
        mxc_register_device(&mxc_uart_device0, &uart_pdata);
        mxc_register_device(&mxc_uart_device1, &uart_pdata);
+
+       /* SMSC9118 IRQ pin */
+       gpio_direction_input(MX31_PIN_GPIO1_0);
  }

  static void __init armadillo5x0_timer_init(void)
```

Abilitare la compilazione del driver di rete *smsc911x.c* ed impostazione del root filesystem di rete. Per abilitare la compilazione del driver di rete nel menu di configurazione del kernel deve essere selezionata per la presenza statica l'opzione: `Device Drivers --> Network device support --> Ethernet (10 or 100Mbit) --> SMSC LAN911x/LAN921x families embedded ethernet support`.

Mentre per abilitare il supporto a root filesystem di rete attraverso il protocollo NFS devono essere selezionate le seguenti impostazioni:

`Networking support --> Networking options --> TCP/IP networking` e `IP: kernel level autoconfiguration`.

`File systems --> Network File Systems --> NFS client support` e `NFS client support for NFS version 3` e `Root file system on NFS`.

Compilato e scaricato il kernel così modificato, nel menu di Hermit possono essere impostati i parametri relativi al root filesystem di rete:

```
hermit> setenv console=ttymxc0 rootfstype=nfs root=/dev/nfs
   nfsroot=192.168.0.2:/media/dati/android/rootfs ip
   =192.168.0.10:192.168.0.2:192.168.0.2:255.255.255.0:armadillo
   :eth0:off
```

/media/dati/android è l'espansione della variabile `PRJROOT`.

La sintassi e la semantica dei parametri qui scritti sono definite nel documento [13].

Procedendo con l'operazione di boot del sistema, se la board è correttamente collegata alla rete di sviluppo ed i parametri del root filesystem nfs sono corretti, allora nella console seriale verrà alla fine presentata la shell di comando della board data da *BusyBox*.

Wake on Lan? Il controller Lan SMSC 9118 prevede un secondo segnale di interrupt chiamato `PME_IRQ` (*Power Management Event Interrupt*) per implementare il meccanismo *Wake on Lan* nel caso di attività di rete ricevuta mentre il sistema si trova nello stato di risparmio energetico.

Dato che il driver *smsc911x.c* non implementa le funzionalità di Power Management di Linux, questo segnale non è stato considerato.

6.3.4 SDHC MMC

Raggiunta l'interattività del sistema, si procede nel supporto dei dispositivi di piattaforma.

In riferimento alla sezione 5.1.4 la board monta un alloggiamento per schede di memoria SD/MMC collegato al primo *Host Controller* SDHC dell'Application Processor i.MX31 (vedi la sezione4.14) supportato dal driver in kernel */drivers/mmc/host/mxcmmc.c* già abilitato alla compilazione.

Risorse del dispositivo. Essendo un dispositivo di piattaforma nel file */arch/arm/mach-mx3/devices.c* sono presenti i descrittori `mxcsdhc_device0` e `mxcsdhc_device1` per i due controller SDHC completi delle risorse utili, che sono: l'area di memoria dei registri di configurazione dei singoli controller e le singole sorgenti di interrupt per i trasferimenti di dati in DMA.

Parametri del driver. La struttura che determina i possibili parametri del driver *mxc-mmc* è definita nel file `arch/arm/plat-mxc/include/mach/mmc.h`:

Listing 6.15: Estratto del file `arch/arm/plat-mxc/include/mach/mmc.h`.

```
/* board specific SDHC data, optional.
 * If not present, a writable card with 3.3V is assumed.
 */
struct imxmmc_platform_data {
        /* Return values for the get_ro callback should be:
         *   0 for a read/write card
         *   1 for a read-only card
         *   -ENOSYS when not supported (equal to NULL callback)
         *   or a negative errno value when something bad happened
         */
        int (*get_ro)(struct device *);

        /* board specific hook to (de)initialize the SD slot.
         * The board code can call 'handler' on a card detection
         * change giving data as argument.
         */
        int (*init)(struct device *dev, irq_handler_t handler, void *data);
        void (*exit)(struct device *dev, void *data);

        /* available voltages. If not given, assume
         * MMC_VDD_32_33 | MMC_VDD_33_34
         */
        unsigned int ocr_avail;

        /* adjust slot voltage */
        void (*setpower)(struct device *, unsigned int vdd);
};
```

Le due funzioni **`init`** ed **`exit`** permettono di allocare e deallocare dinamicamente le risosrse GPIO e IRQ necessarie alle attività del driver nel suo ciclo di vita, che sono:

Gestione dell'evento `card-detect`: L'alloggiamento montato possiede un semplice circuito elettrico capace di segnalare la presenza o meno di una scheda di memoria SD/MMC.
Il segnale generato è connesso al pin `MX31_PIN_ATA_DMACK` dell'Application Processor e dovrà essere gestito come sorgente di interrupt (per i due fronti: di salita e di discesa del segnale) attraverso la routine **`handler(data)`**.

Analisi dello stato `write-protect` della scheda: Le schede di memoria SD/MMC possiedono un selettore che se attivato preclude la capacità di scrittura nella scheda. Lo stato di questo selettore è analizzabile attraverso lo stato elettrico del

connettore `WP_SW` collegato al pin `MX31_PIN_ATA_RESET_B` dell'Application Processor.

La funzione **`get_ro`** infine deve analizzare lo stato del segnale `WP_SW` per fornire le condizioni attuali del selettore `write-protect` della scheda.

I rimanenti parametri non sono stati presi in considerazione. Riguardano modifiche al comportamento elettrico del controller non necessarie per la configurazione circuitale presente nella board.

Modifiche al codice di supporto. Di seguito sono riportate le modifiche al file *armadillo5x0.c* compreso l'allocazione dei pin di interfaccia fisica del primo controller SDHC.

Listing 6.16: armadillo5x0-03ToSDHC.patch

```
***************
*** 34,39 ****
--- 34,40 ----
  #include <mach/imx-uart.h>
  #include <mach/iomux-mx3.h>
  #include <mach/board-armadillo5x0.h>
+ #include <mach/mmc.h>

  #include "devices.h"

***************
*** 50,55 ****
--- 51,126 ----
        MX31_PIN_RXD2__RXD2,
        /* LAN9118_IRQ */
        IOMUX_MODE(MX31_PIN_GPIO1_0, IOMUX_CONFIG_GPIO),
+       /* SDHC1 */
+       MX31_PIN_SD1_DATA3__SD1_DATA3,
+       MX31_PIN_SD1_DATA2__SD1_DATA2,
+       MX31_PIN_SD1_DATA1__SD1_DATA1,
+       MX31_PIN_SD1_DATA0__SD1_DATA0,
+       MX31_PIN_SD1_CLK__SD1_CLK,
+       MX31_PIN_SD1_CMD__SD1_CMD,
+ };
+
+ /*
+  * SDHC 1
+  * MMC support
+  */
+ static int armadillo5x0_sdhc1_get_ro(struct device *dev)
+ {
+       return gpio_get_value(IOMUX_TO_GPIO(MX31_PIN_ATA_RESET_B));
+ }
+
+ static int armadillo5x0_sdhc1_init(struct device *dev,
+                                    irq_handler_t detect_irq, void *data)
```

```
+ {
+       int ret;
+       int gpio_det, gpio_wp;
+
+       gpio_det = IOMUX_TO_GPIO(MX31_PIN_ATA_DMACK);
+       gpio_wp = IOMUX_TO_GPIO(MX31_PIN_ATA_RESET_B);
+
+       ret = gpio_request(gpio_det, "sdhc-card-detect");
+       if (ret)
+               return ret;
+
+       gpio_direction_input(gpio_det);
+
+       ret = gpio_request(gpio_wp, "sdhc-write-protect");
+       if (ret)
+               goto err_gpio_free;
+
+       gpio_direction_input(gpio_wp);
+
+       /* When supported the trigger type have to be BOTH */
+       ret = request_irq(IOMUX_TO_IRQ(MX31_PIN_ATA_DMACK), detect_irq,
+                         IRQF_DISABLED | IRQF_TRIGGER_FALLING,
+                         "sdhc-detect", data);
+
+       if (ret)
+               goto err_gpio_free_2;
+
+       return 0;
+
+ err_gpio_free_2:
+       gpio_free(gpio_wp);
+
+ err_gpio_free:
+       gpio_free(gpio_det);
+
+       return ret;
+
+ }
+
+ static void armadillo5x0_sdhc1_exit(struct device *dev, void *data)
+ {
+       free_irq(IOMUX_TO_IRQ(MX31_PIN_ATA_DMACK), data);
+       gpio_free(IOMUX_TO_GPIO(MX31_PIN_ATA_DMACK));
+       gpio_free(IOMUX_TO_GPIO(MX31_PIN_ATA_RESET_B));
+ }
+
+ static struct imxmmc_platform_data sdhc_pdata = {
+       .get_ro = armadillo5x0_sdhc1_get_ro,
+       .init = armadillo5x0_sdhc1_init,
+       .exit = armadillo5x0_sdhc1_exit,
  };

  /*
***************
*** 109,114 ****
--- 180,188 ----
```

```
        /* SMSC9118 IRQ pin */
        gpio_direction_input(MX31_PIN_GPIO1_0);
+
+       /* Register SDHC */
+       mxc_register_device(&mxcsdhc_device0, &sdhc_pdata);
  }

  static void __init armadillo5x0_timer_init(void)
```

Utilizzare il driver *mxcmmc.c*. Compilato e scaricato nella board Armadillo 500 il kernel con le modifiche apportate, se inserita una scheda di memoria SD card, il driver provvederà a creare la struttura di block devices così organizzata:

```
MMC block devices:
                /dev/mmcblk0      SD/MMC card 1
                /dev/mmcblk0p1    Partizione 1 nella MMC card 1
                    ...
                /dev/mmcblk1      SD/MMC card 2
                    ...
```

Problemi nella gestione degli eventi di inserzione ed uscita di una scheda SD/MMC. Nella funzione **`armadillo5x0_sdhc1_init`** veiene inizializzato l'*interrupt handler* necessario a reagire agli eventi di inserzione ed uscita di una scheda SD/MMC[34]. La funzione **`handler`** è stata scritta per gestire entrambi gli eventi, differenziati solamente dallo stato interno del driver che vuole l'evento di uscita successivo all'evento di inserzione. Perché sia consistente questo comportamento, la sorgente di interrupt proveniente dal pin `MX31_PIN_ATA_DMACK` deve essere impostata per generare interruzioni sia nel fronte di salita che nel fronte di discesa del segnale.

Al momento della scrittura del codice di supporto, la piattaforma `plat-mxc` non supportava la modalità di generazione delle interruzioni descritta dato che lo stesso i.MX31 non la supporta, risultando in un comportamento errato di generazione degli eventi. E' presente nella coda delle patch una modifica apposita che supporterebbe tale comportamento con un semplice work-araund.

[34] In riferimento sempre al documento [7] per la descrizione dei metodi utilizzati.

6.3.5 Output Video

In riferimento alla sezione 5.1.5 nella board Armadillo 500 l'interfaccia Display Interface è stata connessa ad un modulo Digital to Analog Converter ADV7125 della Analog Devices. Questo permette di pilotare un normale schermo con ingresso VGA analogico utilizzando l'output del Synchronous Display Controller interno all'Application Processor i.MX31.

Nel kernel tree è presente il driver */drivers/video/mx3fb.c* che implementa le API di interfaccia Linux Frame Buffer[6] configurando la catena interna di output video del modulo IPU per utilizzare il controller SDC.

Il trasferimento dati dal Frame Buffer in memoria centrale, alla memoria video del controller avviene tramite il sottosistema DMA del modulo IPU gestito dal driver */drivers/dma/ipu/ipu_idmac.c*.

I segnali dell'interfaccia Display Interface. I segnali utili alla configurazione scelta per la catena di output IPU sono i seguenti [33, tabella 44-12]:

Identificatore pin	Funzione	Descrizione
MX31_PIN_VSYNC3	DISPB_D3_VSYNC	Sincronizzazione di Frame.
MX31_PIN_HSYNC	DISPB_D3_HSYNC	Sincronizzazione di riga.
MX31_PIN_FPSHIFT	DISPB_D3_CLK	Clock dei dati.
MX31_PIN_LD[0:17]	DISPB_DATA[0:17]	Output digitale dei dati.
MX31_PIN_DRDY0	DISPB_D3_DRDY	Data enable.
MX31_PIN_LCS1	GPIO	Segnale Power Save.

Sono presenti 18 linee di output per la definizione del colore del singolo pixel in configurazione 6:6:6 (6 bit per colore primario RGB) collegate ai rispettivi pin MSB del modulo DAC (che prevede un'interfaccia digitale in ingresso a 24 bit in configurazione 8:8:8).

Il segnale Data Ready (DRDY) indica la validità o meno dei valori delle linee di output ed è connesso al pin BLANK del modulo DAC di modo da rendere nulla l'uscita analogica quando i dati digitali non sono validi. Questo meccanismo viene utilizzato ad alto livello per spegnere lo schermo in regime di Power Save.

Il segnale LCS1 è connesso al pin PSAVE del modulo DAC. Quindi può essere utilizzato per aumentare l'effetto di risparmio in regime di Power

Save.

I segnali di sincronizzazione VSYNC e HSYNC vengono riportati direttamente nel connettore analogico tramite un circuito driver di canale mentre CLK è collegato in ingresso al modulo DAC e determina, nel suo fronte di salita, l'istante in cui i dati nel bus vengono salvati per la conversione.

Impostazioni temporali. La figura 6.1 schematizza l'andamento temporale dei segnali di sincronizzazione che regolano la trasmissione dati nello standard VGA. Comprenderne il significato è necessario per poi configurare correttamente i parametri del driver video.

Lo standard VGA utilizza i due segnali VSYNC e HSYNC per sincronizzare i circuiti di visualizzazione interni allo schermo con i dati presenti nel canale: lo standard VGA nasce a supporto dei vecchi schermi analogici CRT dove il disegno del frame a schermo avviene sequenzialmente per punti di ogni riga (*pixel*) per poi ricominciare all'inizio della riga successiva (*retrace orizzontale*) ed al termine del frame ricominciare all'inizio di quello successivo (*retrace verticale*). Le operazioni di retrace richiedono un tempo non nullo al quale corrisponde la durata degli *impulsi* dei due segnali HSYNC (t_{hs}) e VSYNC (t_{vs}) che espresse in pixel determinano l'*Area di sincronismo* in figura 6.1.

Oltre ai tempi di sincronizzazione, negli schermi CRT l'*Area di disegno* è più grande dell'*Area visualizzata* effettivamente. Questo per due motivi: uno elettrico di "inseguimento" del segnale analogico dei dati (i circuiti interni che traducolo il segnale analogico in colore a schermo necessitano di un determinato tempo per colmare la differenza tra lo stato dell'ultimo pixel di riga ed il primo della successiva) ed uno logico che determina il posizionamento del frame nello schermo, il quale può uscire dall'*Area visualizzata*. Per posizionare correttamente il frame nell'area visibile e non distorta cromaticamente si può intervenire nei tempi di traslazione t_{left}, t_{right},t_{up},t_{down} che in unità pixel sono chiamati anche *margini* del frame.

Tutte queste grandezze unite al tempo utile dei dati visualizzati a schermo sono legate dalle seguenti equazioni:

$$\frac{1}{F_{frame}} \cong T_{vs} = t_{up} + t_{row} \cdot n_{rows} + t_{down};$$

$$t_{row} = t_{left} + n_{pixels} \cdot t_{pixel} + t_{right};$$

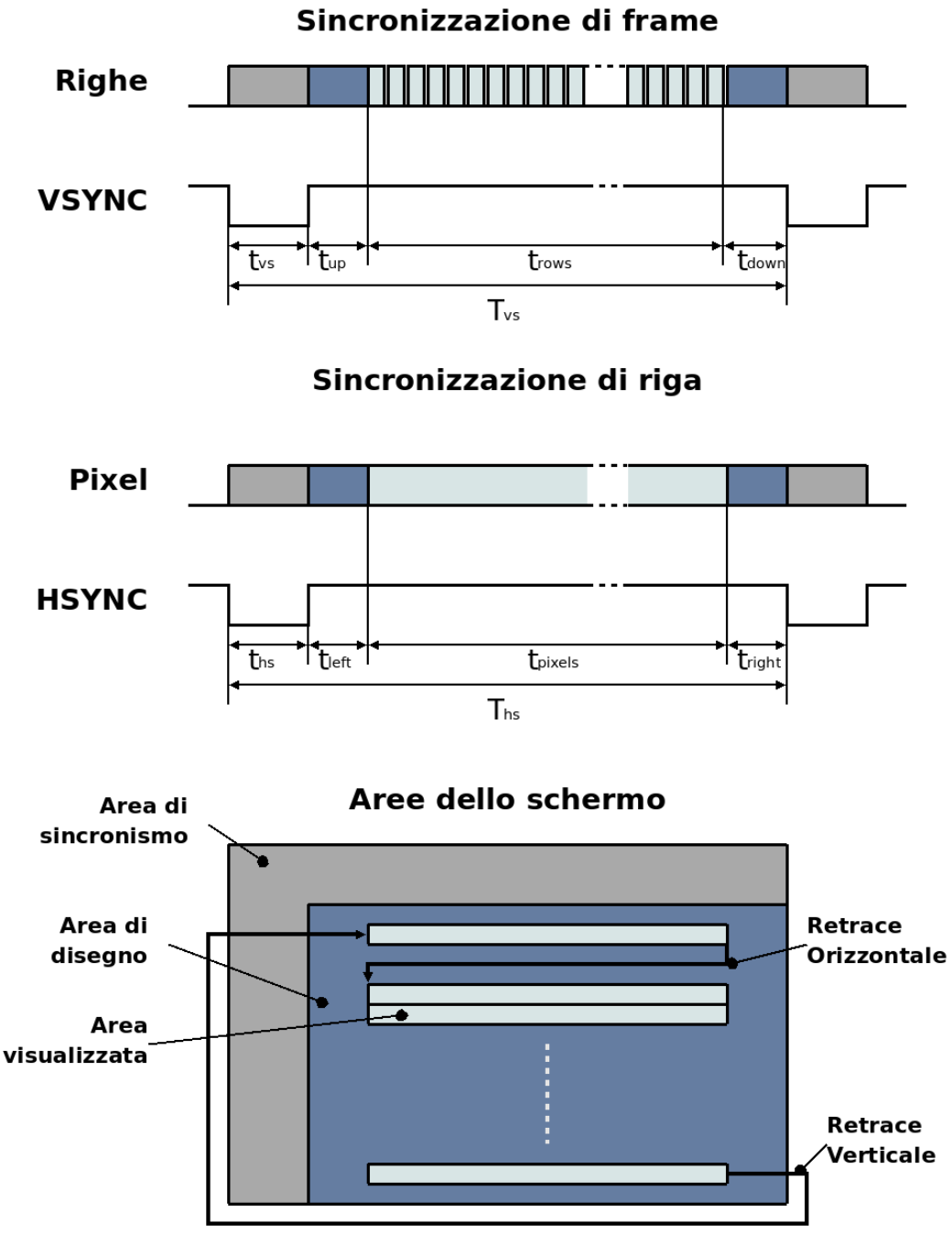


Figura 6.1: Andamento temporale dei segnali di sincronizzazione nello standard VGA.

Dove F_{frame}(Frequenza di aggiornamento del frame), t_{row} (Periodo di aggiornamento di riga) e t_{pixel} (Periodo di pixel) sono standardizzati per le differenti risoluzioni video esistenti.

Parametri del driver*mx3fb.c*. La struttura che determina i possibili parametri del driver *mx3fb* è definita nel file header seguente:

Listing 6.17: *arch/arm/plat-mxc/include/mach/mx3fb.h*.

```
#ifndef __ASM_ARCH_MX3FB_H__
#define __ASM_ARCH_MX3FB_H__

#include <linux/device.h>
#include <linux/fb.h>

/* Proprietary FB_SYNC_ flags */
#define FB_SYNC_OE_ACT_HIGH     0x80000000
#define FB_SYNC_CLK_INVERT      0x40000000
#define FB_SYNC_DATA_INVERT     0x20000000
#define FB_SYNC_CLK_IDLE_EN     0x10000000
#define FB_SYNC_SHARP_MODE      0x08000000
#define FB_SYNC_SWAP_RGB        0x04000000
#define FB_SYNC_CLK_SEL_EN      0x02000000

/**
 * struct mx3fb_platform_data - mx3fb platform data
 *
 * @dma_dev:    pointer to the dma-device, used for dma-slave connection
 * @mode:       pointer to a platform-provided per mxc_register_fb() videomode
 */
struct mx3fb_platform_data {
        struct device                   *dma_dev;
        const char                      *name;
        const struct fb_videomode       *mode;
        int                             num_modes;
};

#endif
```

Il device `dma_dev` verrà impostato al device risultante dalla registrazione del dispositivo di piattaforma `mx3_ipu`.

`mode` è un array di `num_modes` strutture `fb_videomode` che indica le caratteristiche delle modalità video supportate per i dispositivi display collegabili alla board, delle quali viene impostata di default quella identificata dal parametro `name`.

La struttura `fb_videomode` è così definita:

```
struct fb_videomode {
        const char *name;       /* mode name */
```

```
        u32 refresh;            /* optional */
        u32 xres;               /* x frame resolution */
        u32 yres;               /* x frame resolution */
        u32 pixclock;           /* pixel period in pico */
        u32 left_margin;        /* H margins in pixels */
        u32 right_margin;
        u32 upper_margin;       /* V margins in rows */
        u32 lower_margin;
        u32 hsync_len;          /* in pixels */
        u32 vsync_len;          /* in rows */
        u32 sync;               /* Proprietary sync mode */
        u32 vmode;              /* FB video mode */
        u32 flag;               /* FB flags */
};
```

Si è voluto supportare i due standard di risoluzione: VGA (640 x 480 @ 60Hz) e SVGA (800 x 600 @ 56Hz) per i quali sono stati definiti i seguenti parametri:

	$F_{pixel}[MHz]$	$F_{row}[Hz]$	$F_{frame}[Hz]$	Pixel totali per riga	Righe totali per frame
VGA	25,175	31469	59,94	800	525
SVGA	33,33	35161	56,34	948	624

Le dimensioni dei margini e la lunghezza degli impulsi dei segnali `VSYNC` e `HSYNC` sono stati determinati in modo empirico testandone i risultati.

Modifiche al codice di supporto. Di seguito sono riportate le modifiche al file *armadillo5x0.c*:

Listing 6.18: armadillo5x0-04ToFB.patch

```
***************
*** 35,40 ****
--- 35,42 ----
  #include <mach/iomux-mx3.h>
  #include <mach/board-armadillo5x0.h>
  #include <mach/mmc.h>
+ #include <mach/ipu.h>
+ #include <mach/mx3fb.h>

  #include "devices.h"

***************
*** 58,63 ****
--- 60,139 ----
        MX31_PIN_SD1_DATA0__SD1_DATA0,
```

```
        MX31_PIN_SD1_CLK__SD1_CLK,
        MX31_PIN_SD1_CMD__SD1_CMD,
+       /* Framebuffer */
+       MX31_PIN_LD0__LD0,
+       MX31_PIN_LD1__LD1,
+       MX31_PIN_LD2__LD2,
+       MX31_PIN_LD3__LD3,
+       MX31_PIN_LD4__LD4,
+       MX31_PIN_LD5__LD5,
+       MX31_PIN_LD6__LD6,
+       MX31_PIN_LD7__LD7,
+       MX31_PIN_LD8__LD8,
+       MX31_PIN_LD9__LD9,
+       MX31_PIN_LD10__LD10,
+       MX31_PIN_LD11__LD11,
+       MX31_PIN_LD12__LD12,
+       MX31_PIN_LD13__LD13,
+       MX31_PIN_LD14__LD14,
+       MX31_PIN_LD15__LD15,
+       MX31_PIN_LD16__LD16,
+       MX31_PIN_LD17__LD17,
+       MX31_PIN_VSYNC3__VSYNC3,
+       MX31_PIN_HSYNC__HSYNC,
+       MX31_PIN_FPSHIFT__FPSHIFT,
+       MX31_PIN_DRDY0__DRDY0,
+       IOMUX_MODE(MX31_PIN_LCS1, IOMUX_CONFIG_GPIO), /*ADV7125_PSAVE*/
+ };
+
+ /*
+  * FB support
+  */
+ static const struct fb_videomode fb_modedb[] = {
+       {       /* 640x480 @ 60 Hz */
+               .name           = "CRT-VGA",
+               .refresh        = 60,
+               .xres           = 640,
+               .yres           = 480,
+               .pixclock       = 39721,
+               .left_margin    = 35,
+               .right_margin   = 115,
+               .upper_margin   = 43,
+               .lower_margin   = 1,
+               .hsync_len      = 10,
+               .vsync_len      = 1,
+               .sync           = FB_SYNC_OE_ACT_HIGH,
+               .vmode          = FB_VMODE_NONINTERLACED,
+               .flag           = 0,
+       }, {/* 800x600 @ 56 Hz */
+               .name           = "CRT-SVGA",
+               .refresh        = 56,
+               .xres           = 800,
+               .yres           = 600,
+               .pixclock       = 30000,
+               .left_margin    = 30,
+               .right_margin   = 108,
+               .upper_margin   = 13,
```

```
+                .lower_margin   = 10,
+                .hsync_len      = 10,
+                .vsync_len      = 1,
+                .sync           = FB_SYNC_OE_ACT_HIGH | FB_SYNC_HOR_HIGH_ACT |
+                                  FB_SYNC_VERT_HIGH_ACT,
+                .vmode          = FB_VMODE_NONINTERLACED,
+                .flag           = 0,
+        },
+ };
+
+ static struct ipu_platform_data mx3_ipu_data = {
+        .irq_base = MXC_IPU_IRQ_START,
+ };
+
+ static struct mx3fb_platform_data mx3fb_pdata = {
+        .dma_dev        = &mx3_ipu.dev,
+        .name           = "CRT-VGA",
+        .mode           = fb_modedb,
+        .num_modes      = ARRAY_SIZE(fb_modedb),
  };

  /*
***************
*** 183,188 ****
--- 259,268 ----

        /* Register SDHC */
        mxc_register_device(&mxcsdhc_device0, &sdhc_pdata);
+
+       /* Register FB */
+       mxc_register_device(&mx3_ipu, &mx3_ipu_data);
+       mxc_register_device(&mx3_fb, &mx3fb_pdata);
  }

  static void __init armadillo5x0_timer_init(void)
```

Utilizzare il driver *mx3fb.c*. Compilato e scaricato nella board Armadillo 500 il kernel con le modifiche apportate (il driver *mx3fb.c* è già stato abilitato dalla configurazione di piattaforma per l'inserimento statico nell'immagine del kernel) con la seguente riga di parametri del kernel è possibile abilitare nella fase di boot l'output video:

```
hermit> setenv video=mx3fb:CRT-VGA,bpp=16 console=ttymxc0
    rootfstype=nfs root=/dev/nfs nfsroot=192.168.0.2:/media/dati/
    android/rootfs ip
    =192.168.0.10:192.168.0.2:192.168.0.2:255.255.255.0:armadillo
    :eth0:off
```

Il parametro `video=` definisce quale driver video abilitare, con che modalità e la profondità di colore da adottare.

Collegato uno schermo al connettore D-Sub15 è possibile eseguire dei test di funzionalità utilizzando le utility compilate appositamente (vedi la sezione 6.2.6) che sono:

`target$ fbtest` : Mostra a schermo una serie di test per verificare la posizione del frame, la geometria ed i colori visualizzati.

`target$ fbv nomefile` : Visualizza a schermo l'immagine con nome `nomefile`.

Problemi di resa dei colori delle immagini visualizzate. I test effettuati hanno evidenziato un problema nella resa dei colori di determinati pixel sullo schermo. L'analisi a confronto del codice sorgente del driver *mx3fb.c* e del corrispondente driver *mxcfb.c* presente nella versione Atmark 2.6.26 del kernel ha evidenziato la seguente differenza risolutiva:

Listing 6.19: Adattamento della forma d'onda del clock in ingresso al modulo ADV7125 nel driver *mx3fb.c*.

```
Subject: [PATCH] Armadillo 500 pixel writing timing adaptation.

Signed-off-by: Alberto Panizzo <maramaopercheseimorto@gmail.com>
---
 drivers/video/mx3fb.c |   12 ++++++++++-
 1 files changed, 11 insertions(+), 1 deletions(-)

diff --git a/drivers/video/mx3fb.c b/drivers/video/mx3fb.c
index 054ef29..e4a3115 100644
--- a/drivers/video/mx3fb.c
+++ b/drivers/video/mx3fb.c
@@ -33,6 +33,7 @@

 #include <asm/io.h>
 #include <asm/uaccess.h>
+#include <asm/mach-types.h>

 #define MX3FB_NAME              "mx3_sdc_fb"

@@ -515,7 +516,16 @@ static int sdc_init_panel(struct mx3fb_data *mx3fb, enum
    ipu_panel panel,
         * fewer. Subtract 1 extra from DISP3_IF_CLK_DOWN_WR based on timing
         * debug. DISP3_IF_CLK_UP_WR is 0
         */
-       mx3fb_write_reg(mx3fb, (((div / 8) - 1) << 22) | div, DI_DISP3_TIME_CONF
    );
+       /*
+        * Due to timing constraints in pixel writing on ADV7125 Video DAC
+        */
+       if (machine_is_armadillo5x0()) {
+               old_conf = (div |
```

```
+                                ((div >> 4) - 1) << 24 | /* DISP3_IF_CLK_DOWN_WR */
+                                ((div >> 4) - 2) << 14); /* DISP3_IF_CLK_UP_WR */
+                mx3fb_write_reg(mx3fb, old_conf, DI_DISP3_TIME_CONF);
+        } else
+                mx3fb_write_reg(mx3fb, (((div / 8) - 1) << 22) | div,
    DI_DISP3_TIME_CONF);

        /* DI settings */
        old_conf = mx3fb_read_reg(mx3fb, DI_DISP_IF_CONF) & 0x78FFFFFF;
--
1.5.4.3
```

Il manuale dell'Application Processor i.MX31 [33, Sezione 44.3.3.8.15] definisce il significato del registro `DI_DISP3_TIME_CONF` che regola la forma d'onda del clock in uscita del terzo display (proprio del controller SDC). `DI_DISP3_TIME_CONF` memorizza i tre valori (dal bit meno significativo):

`DISP3_IF_CLK_PER_WR [11:0]`: Numero a 4 cifre decimali (parte intera bit 11:4, parte decimale bit 3:0).
Divide la frequenza del clock `HSP_CLK` per ottenere la frequenza del segnale `DISPB_D3_CLK`.
Il periodo di pixel quindi è così ottenuto: $t_{pixel} = T_{HSP_CLK} \cdot DISP3_IF_CLK_PER_WR$.

`DISP3_IF_CLK_UP_WR [21:12]`: Numero a 2 cifre decimali (parte intera bit 21:14, parte decimale bit 13:12).
Indica la posizione del fronte di salita del segnale di clock `DISPB_D3_CLK`.
Il fronte di salita si troverà all'istante: $\lceil 2 \cdot DISP3_IF_CLK_UP_WR \cdot T_{HSP_CLK} \rceil / 2$.

`DISP3_IF_CLK_DOWN_WR[31:22]`:Numero a 2 cifre decimali (parte intera bit 31:24, parte decimale bit 23:22).
Indica la posizione del fronte di discesa del segnale di clock `DISPB_D3_CLK`.
Il fronte di discesa si troverà all'istante: $\lceil 2 \cdot DISP3_IF_CLK_DOWN_WR \cdot T_{HSP_CLK} \rceil / 2$ che necessariamente deve essere successivo a quello del fronte di salita.

La varibile div nella funzione **`sdc_init_panel`** è calcolata per la generazione di un segnale di clock in uscita con periodo pari all'impostazione

di modalità `mode.pixclock` nella forma corretta per essere scritta nella porzione `DISP3_IF_CLK_PER_WR` (valore reale con shift a sinistra di 4 bit).

Il comportamento standard del driver allora impone il fronte di salita all'istante iniziale del periodo di clock ed il fronte di discesa circa a metà periodo meno T_{HSP_CLK}.

Dai datasheet del modulo ADV7125 [19] si apprendono le temporizzazioni adottate nel processo di lettura dei dati nel bus, conversione ed output del relativo segnale: come mostrato figura 6.2 il modulo ADV7125 esegue la lettura nel bus (latching nei registri interni) dei dati digitali nel fronte di salita del clock e dopo il tempo di conversione necessario (circa $5ns$) riporta in uscita il segnale analogico per ogni componente di colore.

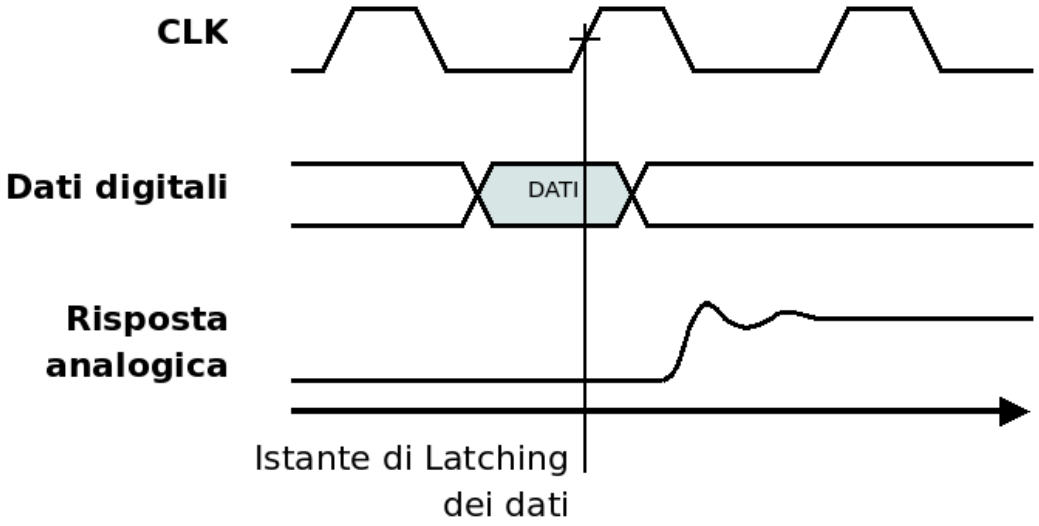


Figura 6.2: Andamento temporale dell'output analogico del modulo DAC rispetto agli input digitali.

Impostare il fronte di salita all'inizio del periodo di clock può portare allora a letture di dati in fase di transizione nel bus e quindi non sempre corrette.

La modifica persentata al driver interviene proprio in questo, spostando l'impulso di clock nella porzione superiore del periodo, imponendo il fronte di salita all'istante $\simeq t_{pixel} - 2 \cdot T_{HSP_CLK}$ ed il fronte di discesa all'istante $\simeq t_{pixel} - T_{HSP_CLK}$.

La patch presentata però non è stata accettata dato che inserisce codice specifico per una singola board all'interno di un driver generico.

Sono state fatte delle proposte per inserire un nuovo parametro di configurazione del driver di modo da esplicitare le posizioni dei fronti di salita e discesa in modo proporzionale alla dimensione del periodo di pixel.

6.3.6 Modulo flash NOR

In riferimento alla sezione 5.1.1 nel modulo CPU Armadillo 500 è presente una memoria flash di tipo NOR organizzata come in tabella 5.1.

Il chip di memoria è connesso attraverso un bus dati a 16 bit [17] ed è mappato direttamente nello spazio di indirizzi fisici dell'Application Processor secondo la tabella Memory Map in appendice A. Il driver generico in kernel */drivers/mtd/physmap.c* fornisce supporto per questa tipologia di mapping delle memorie flash per il quale verrà creato un descrittore di dispositivo specifico.

L'interfaccia di comando delle flash Intel è implementata nel driver di chip */drivers/mtd/chips/cfi_cmdset_0001.c*.

Risorse del dispositivo. Il driver *physmap.c* richiede la sola risorsa di memoria che indica la posizione della memoria flash nello spazio di indizizzi dell'Application Processor. In riferimento alla Memory Map in appendice A:

```
static struct resource armadillo5x0_nor_flash_resource = {
        .flags          = IORESOURCE_MEM,
        .start          = CS0_BASE_ADDR,
        .end            = CS0_BASE_ADDR + SZ_16M - 1,
};
```

Parametri del driver *physmap.c*. I parametri del driver sono definiti dalla struttura `physmap_flash_data`:

```
struct map_info;

struct physmap_flash_data {
        unsigned int            width;          /* In Octets */
        void                    (*set_vpp)(struct map_info *, int);
        unsigned int            nr_parts;
        unsigned int            pfow_base;
        struct mtd_partition    *parts;
};
```

Dove la struttura `mtd_partition` è definita come:

```
/*
 * Partition definition structure:
 *
 * An array of struct partition is passed along with a MTD object to
 * add_mtd_partitions() to create them.
 *
 * For each partition, these fields are available:
 * name: string that will be used to label the partition's MTD device.
 * size: the partition size; if defined as MTDPART_SIZ_FULL, the partition
 *       will extend to the end of the master MTD device.
 * offset: absolute starting position within the master MTD device; if
 *       defined as MTDPART_OFS_APPEND, the partition will start where the
 *       previous one ended; if MTDPART_OFS_NXTBLK, at the next erase block.
 * mask_flags: contains flags that have to be masked (removed) from the
 *       master MTD flag set for the corresponding MTD partition.
 *       For example, to force a read-only partition, simply adding
 *       MTD_WRITEABLE to the mask_flags will do the trick.
 *
 * Note: writeable partitions require their size and offset be
 * erasesize aligned (e.g. use MTDPART_OFS_NEXTBLK).
 */

struct mtd_partition {
        char *name;                     /* identifier string */
        uint64_t size;                  /* partition size */
        uint64_t offset;                /* offset within the master MTD space */
        uint32_t mask_flags;            /* master MTD flags to mask out for this
                 partition */
        struct nand_ecclayout *ecclayout;       /* out of band layout for this
                partition (NAND only)*/
        struct mtd_info **mtdp;         /* pointer to store the MTD object */
};

#define MTDPART_OFS_NXTBLK      (-2)
#define MTDPART_OFS_APPEND      (-1)
#define MTDPART_SIZ_FULL        (0)
```

Dato che il bus di collegamento è a 16 bit, il parametro `width` sarà impostato a 2.

La definizione della mappa delle partizioni invece sarà impostata di modo da riflettere esattamente l'organizzazione interna della flash.

Modifiche al codice di supporto. Di seguito sono riportate le modifiche al file *armadillo5x0.c*.

Listing 6.20: armadillo5x0-05ToNOR.patch

```
***************
*** 22,27 ****
```

```
--- 22,28 ----
  #include <linux/smsc911x.h>
  #include <linux/interrupt.h>
  #include <linux/irq.h>
+ #include <linux/mtd/physmap.h>

  #include <mach/hardware.h>
  #include <asm/mach-types.h>
***************
*** 87,92 ****
--- 88,135 ----
  };

  /*
+  * MTD NOR Flash
+  */
+ static struct mtd_partition armadillo5x0_nor_flash_partitions[] = {
+       {
+               .name           = "nor.bootloader",
+               .offset         = 0x00000000,
+               .size           = 4*32*1024,
+       }, {
+               .name           = "nor.kernel",
+               .offset         = MTDPART_OFS_APPEND,
+               .size           = 16*128*1024,
+       }, {
+               .name           = "nor.userland",
+               .offset         = MTDPART_OFS_APPEND,
+               .size           = 110*128*1024,
+       }, {
+               .name           = "nor.config",
+               .offset         = MTDPART_OFS_APPEND,
+               .size           = 1*128*1024,
+       },
+ };
+
+ static struct physmap_flash_data armadillo5x0_nor_flash_pdata = {
+       .width          = 2,
+       .parts          = armadillo5x0_nor_flash_partitions,
+       .nr_parts       = ARRAY_SIZE(armadillo5x0_nor_flash_partitions),
+ };
+
+ static struct resource armadillo5x0_nor_flash_resource = {
+       .flags          = IORESOURCE_MEM,
+       .start          = CS0_BASE_ADDR,
+       .end            = CS0_BASE_ADDR + SZ_64M - 1,
+ };
+
+ static struct platform_device armadillo5x0_nor_flash = {
+       .name                   = "physmap-flash",
+       .id                     = -1,
+       .num_resources          = 1,
+       .resource               = &armadillo5x0_nor_flash_resource,
+ };
+
+ /*
```

```
    * FB support
    */
   static const struct fb_videomode fb_modedb[] = {
***************
*** 262,267 ****
--- 305,314 ----
        /* Register FB */
        mxc_register_device(&mx3_ipu, &mx3_ipu_data);
        mxc_register_device(&mx3_fb, &mx3fb_pdata);
+
+       /* Register NOR Flash */
+       mxc_register_device(&armadillo5x0_nor_flash,
+                           &armadillo5x0_nor_flash_pdata);
   }

   static void __init armadillo5x0_timer_init(void)
```

Configurazione del kernel. Il driver *physmap.c* è già abilitato per l'inserimento statico nell'immagine del kernel con il supporto alle partizioni. Ciò che manca è il driver specifico per chip Intel, selezionabile nel menu di configurazione con: `Device Drivers --> Memory Technology Device (MTD) support --> RAM/ROM/Flash chip drivers --> <*> Support for Intel/Sharp flash chips .`

Utilizzare il driver *physmap.c*. Compilato e scaricato nella board Armadillo 500 il kernel con le modifiche apportate il driver *physmap.c*, attraverso i comandi specifici della flash Intel, provvederà a creare la struttura di block devices così organizzata:

```
Memory Technology Device (Flash)
                /dev/mtd0       nor.bootloader (rw)
                /dev/mtd0ro     nor.bootloader (ro)
                /dev/mtd1       nor.kernel (rw)
                /dev/mtd1ro     nor.kernel (ro)
                /dev/mtd2       nor.userland (rw)
                /dev/mtd2ro     nor.userland (ro)
                /dev/mtd3       nor.config (rw)
                /dev/mtd3ro     nor.config (ro)
```

6.3.7 Modulo flash NAND

In riferimento alla sezione 5.1.6 nello strato inferiore della board Armadillo 500 è montato un chip di memoria flash di tipo NAND da 256 MByte

organizzato in pagine da 2 KByte ciascuna e connesso al Controller NAND Flash dell'Application Processor.

I chip flash NAND permettono l'accesso ai dati solamente una pagina alla volta (lettura, scrittura e cancellazione). Per questo non possono essere mappati direttamente in memoria e devono essere connessi ad un modulo controller dedicato.

Per il Controller NAND Flash degli Application Processor della famiglia i.MX è stato sviluppato il driver presente nel kernel tree: */drivers/mtd/nand/mxc_nand.c*, modificato di recente[35] per il supporto a chip flash NAND organizzati in pagine da 2 KByte.

Parametri del driver. La struttura che determina i possibili parametri del driver *mxc_nand* è definita nel file */arch/arm/plat-mxc/include/mach/mxc_nand.h*:

Listing 6.21: Estratto del file */arch/arm/plat-mxc/include/mach/mxc_nand.h*.

```
struct mxc_nand_platform_data {
        int width;      /* data bus width in bytes */
        int hw_ecc;     /* 0 if supress hardware ECC */
};
```

La larghezza del bus dati di connessione è di 8 bit per cui il parametro `width` sarà impostato a 1 e dato che il Controller NAND Flash dell'Application Processor i.MX31 contiene il modulo Hardware Error Code Correction [33, cap 20.8.4], anche il parametro `hw_ecc` sarà impostato a 1.

Extra configurazione del Controller NAND Flash. Il Controller NAND Flash utilizza il bit di stato `NFMS` presente nel registro `MXC_CCM_RCSR` per venire a conoscenza della dimensione delle pagine del chip flash collegato (0: 512-byte, 1: 2Kbyte). Questo bit dovrebbe essere correttamente inizializzato durante il processo di boot a 1 dal boot loader, cosa che non avviene.

A questo si è posto rimedio nella funzione di init della piattaforma armadillo5x0, utilizzando le funzioni di i/o fornite dall'interfaccia */linux/io.h* e la definizione locale degli indirizzi dei registri di configurazione presente nel file */arch/arm/mach-mx3/crm_regs.h*.

[35] Dalla commit bd3fd62ecc99c709739cb969be76f44903a4043b mtd: MXC NAND support for 2KiB page size flashes

Modifiche al codice di supporto. Di seguito sono riportate le modifiche al file *armadillo5x0.c*.

Listing 6.22: armadillo5x0-06ToNAND.patch

```
***************
*** 23,28 ****
--- 23,29 ----
  #include <linux/interrupt.h>
  #include <linux/irq.h>
  #include <linux/mtd/physmap.h>
+ #include <linux/io.h>

  #include <mach/hardware.h>
  #include <asm/mach-types.h>
***************
*** 38,45 ****
--- 39,48 ----
  #include <mach/mmc.h>
  #include <mach/ipu.h>
  #include <mach/mx3fb.h>
+ #include <mach/mxc_nand.h>

  #include "devices.h"
+ #include "crm_regs.h"

  static int armadillo5x0_pins[] = {
        /* UART1 */
***************
*** 88,93 ****
--- 91,104 ----
  };

  /*
+  * NAND Flash
+  */
+ static struct mxc_nand_platform_data armadillo5x0_nand_flash_pdata = {
+       .width          = 1,
+       .hw_ecc         = 1,
+ };
+
+ /*
   * MTD NOR Flash
   */
  static struct mtd_partition armadillo5x0_nor_flash_partitions[] = {
***************
*** 309,314 ****
--- 320,331 ----
        /* Register NOR Flash */
        mxc_register_device(&armadillo5x0_nor_flash,
                            &armadillo5x0_nor_flash_pdata);
+
+       /* Register NAND Flash */
+       mxc_register_device(&mxc_nand_device, &armadillo5x0_nand_flash_pdata);
+
+       /* set NAND page size to 2k if not configured via boot mode pins */
```

```
+       __raw_writel(__raw_readl(MXC_CCM_RCSR) | (1 << 30), MXC_CCM_RCSR);
  }

  static void __init armadillo5x0_timer_init(void)
```

Utilizzare il driver *mxc_nand.c*. Compilato e scaricato nella board Armadillo 500 il kernel con le modifiche apportate il driver *mxc_nand.c* provvederà ad aggiungere un device MTD alla struttura di block devices per dispositivi flash:

```
Memory Technology Device (Flash)
                ...
                /dev/mtd3       nor.config (rw)
                /dev/mtd3ro     nor.config (ro)
                /dev/mtd4       NAND (rw)
                /dev/mtd4ro     NAND (ro)
```

6.3.8 GPIO Keyboard

In riferimento alla sezione 5.1.8 nella board Armadillo 500 sono presenti due tasti *low-active* connessi ai due pin gpio GPIO3_3 e GPIO3_2 dell'Application Processor attraverso un semplice circuito RC di debouncing.

Per collegare questi tasti al sottosistema di input Linux esiste il driver in kernel apposito: */drivers/input/keyboard/gpio_keys.c*.

Parametri del driver. La struttura che determina i possibili parametri del driver *gpio-keys* è definita nel file *include/linux/gpio_keys.h*:

Listing 6.23: Estratto del file *include/linux/gpio_keys.h*.

```
struct gpio_keys_button {
        /* Configuration parameters */
        int code;               /* input event code (KEY_*, SW_*) */
        int gpio;
        int active_low;
        char *desc;
        int type;               /* input event type (EV_KEY default, EV_SW) */
        int wakeup;             /* configure the button as a wake-up source */
        int debounce_interval;  /* debounce ticks interval in msecs */
};

struct gpio_keys_platform_data {
        struct gpio_keys_button *buttons;
        int nbuttons;
```

```
        unsigned int rep:1;             /* enable input subsystem auto repeat */
};
```

La struttura `gpio_keys_platform_data` contiene un array di `nbuttons` descrittori `gpio_keys_button`, contenenti a loro volta la definizione completa per ogni tasto che si vuole configurare.

La configurazione scelta collega i due tasti on board agli eventi `EV_KEY`: `KEY_ENTER` e `KEY_BACK` (utili in seguito per la configurazione del sistema di input Android), tutti e due di tipo `wakeup` (La pressione di uno di questi tasti in modalità Power Save, risveglia il sistema).

Modifiche al codice di supporto. Di seguito sono riportate le modifiche al file *armadillo5x0.c*.

Listing 6.24: armadillo5x0-07ToGPIOKey.patch

```
***************
*** 24,29 ****
--- 24,31 ----
  #include <linux/irq.h>
  #include <linux/mtd/physmap.h>
  #include <linux/io.h>
+ #include <linux/input.h>
+ #include <linux/gpio_keys.h>

  #include <mach/hardware.h>
  #include <asm/mach-types.h>
***************
*** 90,95 ****
--- 92,128 ----
        IOMUX_MODE(MX31_PIN_LCS1, IOMUX_CONFIG_GPIO), /*ADV7125_PSAVE*/
  };

+ /* GPIO BUTTONS */
+ static struct gpio_keys_button armadillo5x0_buttons[] = {
+       {
+               .code           = KEY_ENTER, /*28*/
+               .gpio           = IOMUX_TO_GPIO(MX31_PIN_SCLK0),
+               .active_low     = 1,
+               .desc           = "menu",
+               .wakeup         = 1,
+       }, {
+               .code           = KEY_BACK, /*158*/
+               .gpio           = IOMUX_TO_GPIO(MX31_PIN_SRST0),
+               .active_low     = 1,
+               .desc           = "back",
+               .wakeup         = 1,
+       }
+ };
+
```

```
+ static struct gpio_keys_platform_data armadillo5x0_button_data = {
+       .buttons        = armadillo5x0_buttons,
+       .nbuttons       = ARRAY_SIZE(armadillo5x0_buttons),
+ };
+
+ static struct platform_device armadillo5x0_button_device = {
+       .name           = "gpio-keys",
+       .id             = -1,
+       .num_resources  = 0,
+       .dev            = {
+               .platform_data  = &armadillo5x0_button_data,
+       }
+ };
+
  /*
   * NAND Flash
   */
***************
*** 291,296 ****
--- 324,330 ----

  static struct platform_device *devices[] __initdata = {
        &armadillo5x0_smc911x_device,
+       &armadillo5x0_button_device,
  };

  /*
```

Configurazione del kernel. Innanzitutto deve essere abilitato il sottosistema di input con: `Device Drivers --> Input device support --> <*> Generic input layer (needed for keyboard, mouse, ...).`

Poi nel sottomenu `[*] Keyboards (NEW) -->` il driver *gpio_keys.c* è abilitato dalla voce `<*> GPIO Buttons`.

6.3.9 RTC

In riferimento alla sezione 5.1.7, al secondo controller I^2C dell'Application Processor è connesso un chip Seiko Instruments S-35390A con funzionalità di Real Time Clock. Il chip SI S-35390A espone una sorgente di interrupt connessa al pin GPIO `GPIO1_20`.

A supporto del controller I^2C interno all'Application Processor è stato sviluppato il driver in kernel */drivers/i2c/busses/i2c-imx.c* già abilitato alla compilazione e per il quale verrà registrato il device di piattaforma `mxc_i2c_device1`; mentre */drivers/rtc/rtc-s35390a.c* implementa le API RTC utilizzando il chip Seiko Instruments S-35390A.

Parametri del driver *i2c-imx.c*. La struttura che determina i possibili parametri del driver *i2c-imx.c* è definita nel file *include/linux/i2c.h*:

Listing 6.25: Estratto del file *include/linux/i2c.h*.

```
/**
 * struct i2c_board_info - template for device creation
 * @type: chip type, to initialize i2c_client.name
 * @flags: to initialize i2c_client.flags
 * @addr: stored in i2c_client.addr
 * @platform_data: stored in i2c_client.dev.platform_data
 * @archdata: copied into i2c_client.dev.archdata
 * @irq: stored in i2c_client.irq
 */
struct i2c_board_info {
        char            type[I2C_NAME_SIZE];
        unsigned short  flags;
        unsigned short  addr;
        void            *platform_data;
        struct dev_archdata     *archdata;
        int             irq;
};

/**
 * This macro initializes essential fields of a struct i2c_board_info,
 * declaring what has been provided on a particular board.  Optional
 * fields (such as associated irq, or device-specific platform_data)
 * are provided using conventional syntax.
 */
#define I2C_BOARD_INFO(dev_type, dev_addr) \
        .type = dev_type, .addr = (dev_addr)
```

Per ogni device collegato ai bus I^2C è necessario esplicitare un descrittore `i2c_board_info` che anrdà registrato nel controller specifico attraverso la funzione:

`int i2c_register_board_info(int busnum, struct i2c_board_info const *info, unsigned n)`

La macro `I2C_BOARD_INFO` aiuta in questo senso, rendendo la definizione più leggibile in sede di enumerazione dei device.

Per il chip Seiko Instruments S-35390A deve essere registrato un dispositivo di tipo `"s35390a"` (id del driver *rtc-s35390a.c*) all'indirizzo `0x30` (si veda il datasheet del dispositivo [1]).

Dato che il driver *rtc-s35390a.c* non effettua le operazioni di GPIO e IRQ request per la sorgente di interrupt associata al chip S-35390A, allora queste andranno effettuate nella funzione **`armadillo5x0_init`** prima di impostare il parametro associato nel descrittore.

Modifiche al codice di supporto. Di seguito sono riportate le modifiche al file *armadillo5x0.c* comprendenti la corretta configurazione delle funzioni dei pin di interfaccia del secondo controller I^2C.

Listing 6.26: armadillo5x0-08ToI2C.patch

```
***************
*** 26,31 ****
--- 26,32 ----
  #include <linux/io.h>
  #include <linux/input.h>
  #include <linux/gpio_keys.h>
+ #include <linux/i2c.h>

  #include <mach/hardware.h>
  #include <asm/mach-types.h>
***************
*** 90,95 ****
--- 91,106 ----
        MX31_PIN_FPSHIFT__FPSHIFT,
        MX31_PIN_DRDY0__DRDY0,
        IOMUX_MODE(MX31_PIN_LCS1, IOMUX_CONFIG_GPIO), /*ADV7125_PSAVE*/
+       /* I2C2 */
+       MX31_PIN_CSPI2_MOSI__SCL,
+       MX31_PIN_CSPI2_MISO__SDA,
+ };
+
+ /* RTC over I2C*/
+ #define ARMADILLO5X0_RTC_GPIO IOMUX_TO_GPIO(MX31_PIN_SRXD4)
+
+ static struct i2c_board_info armadillo5x0_i2c_rtc = {
+       I2C_BOARD_INFO("s35390a", 0x30),
  };

  /* GPIO BUTTONS */
***************
*** 324,329 ****
--- 335,341 ----

  static struct platform_device *devices[] __initdata = {
        &armadillo5x0_smc911x_device,
+       &mxc_i2c_device1,
        &armadillo5x0_button_device,
  };

***************
*** 360,365 ****
--- 372,389 ----

        /* set NAND page size to 2k if not configured via boot mode pins */
        __raw_writel(__raw_readl(MXC_CCM_RCSR) | (1 << 30), MXC_CCM_RCSR);
+
+       /* RTC */
+       /* Get RTC IRQ and register the chip */
+       if (gpio_request(ARMADILLO5X0_RTC_GPIO, "rtc") == 0) {
+               if (gpio_direction_input(ARMADILLO5X0_RTC_GPIO) == 0)
```

```
+                       armadillo5x0_i2c_rtc.irq = gpio_to_irq(
    ARMADILLO5X0_RTC_GPIO);
+               else
+                       gpio_free(ARMADILLO5X0_RTC_GPIO);
+       }
+       if (armadillo5x0_i2c_rtc.irq == 0)
+               pr_warning("armadillo5x0_init: failed to get RTC IRQ\n");
+       i2c_register_board_info(1, &armadillo5x0_i2c_rtc, 1);
  }

  static void __init armadillo5x0_timer_init(void)
```

Configurazione del kernel. Il supporto al sottosistema I^2C ed il driver *i2c-imx.c* sono già abilitati per l'inserimento statico in kernel. Mentre il supporto ai dispositivi Real Time Clock viene abilitato con: `Device Drivers --> <*> Real Time Clock -->` ed successivamente il driver *rtc-s35390a.c* selezionando: `<*> Seiko Instruments S-35390A` .

Utilizzare il modulo RTC. Compilato e scaricato nella board Armadillo 500 il kernel con le modifiche apportate il driver *rtc-s35390a.c* fornirà un'interfaccia di controllo del dispositivo RTC.

Allora con le seguenti verrà impostato l'orologio di sistema e scritta l' impostazione nella memoria del chip RTC:

```
target$ date -s 'YYYY-MM-DD hh:mm[:ss]'
target$ hwclock -w
```

Mentre `hwclock -s` imposterà l'orologio di sistema con la data letta dal chip RTC. Per mostrare la data attuale del chip RTC è sufficiente invocare `hwclock` senza parametri.

6.3.10 USB

Il software in kernel a supporto dei controller USB Host presenti nell'Application Processor i.MX31 non è ad ora divenuto standard.

Se da un lato i tre controller supportano lo standard EHCI (la maggior parte dei registri di controllo dei moduli Controller sono EHCI compatibili) liberando il driver specifico dalle questioni di protocollo e gestione dei dati nella comunicazine USB, deve essere implementato e standardizzato il codice di gestione specifica dei Controller i.MX (compresa la funzionalità OTG) ed il supporto all'interfaccia ULPI verso i transceiver OTG NXP ISP 1504.

USB in Linux

Stack software lato Host. [12] In Linux lo stack software USB Host si presenta come in figura 6.3.

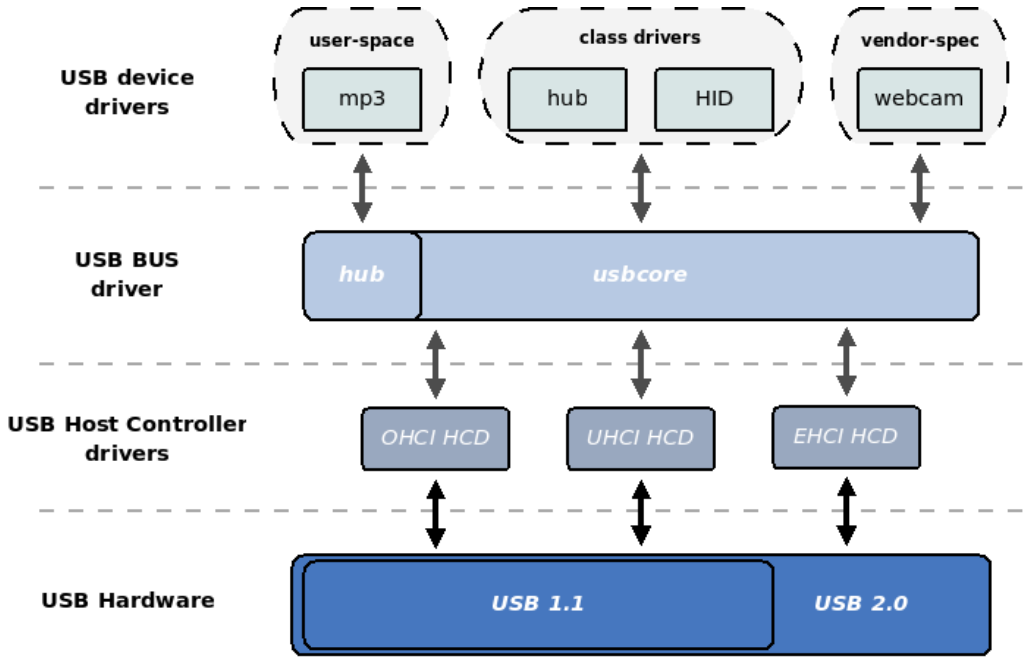


Figura 6.3: Linux USB Host Stack.

usbcore Il modulo "*usbcore*" rappresenta il layer di astrazione per i driver di dispositivo verso le differenti implementazioni dello standard USB. L'interfaccia verso lo strato superiore viene detta *usb driver framework* ed è stata definita in base al modello di descrizione dei dispositivi usb (*usb device model*).

device drivers Moduli software che rendono utilizzabili i dispositivi USB all'utente. I driver USB vengono sviluppati in relazione ad una determinata classe di dispositivi del *device model*, implementando la relativa interfaccia.

usbhcd Driver di gestione del controller hardware. Implementa una specifica interfaccia usb (EHCI OHCI UHCI) per la gestione della comunicazione sul bus USB. Si serve di codice specifico per le operazioni di inizializzazione e gestione del controller hardware installato.

Stack software lato Device. [15] Data la crescente complessità dei sistemi embedded, Linux può essere eseguito sia nelle macchine host ma anche nei dispositivi device.

In Linux lo stack software USB Device si presenta come in figura 6.4.

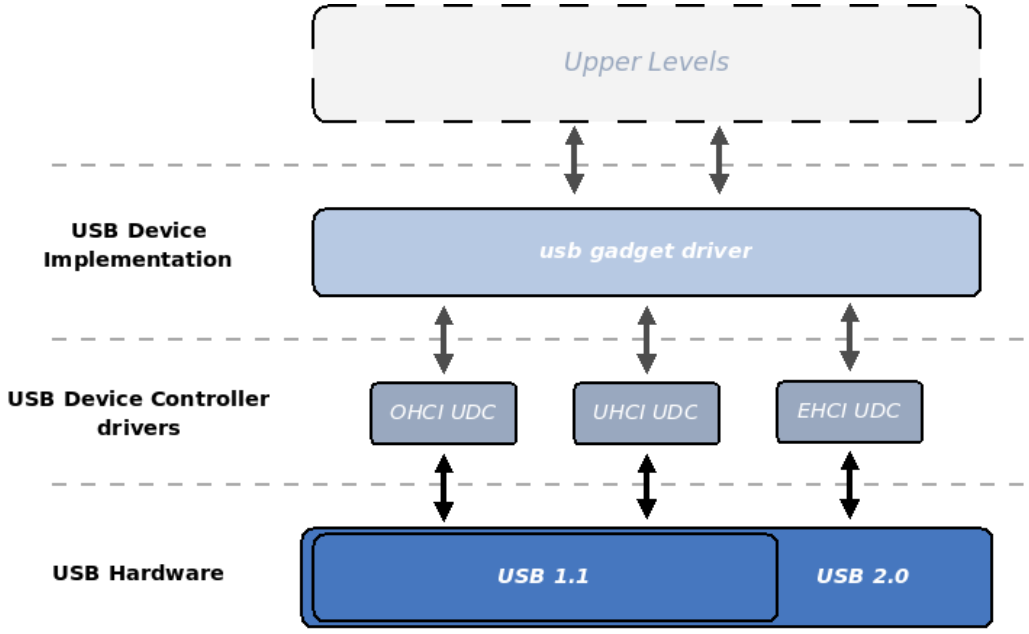


Figura 6.4: Linux USB Device Stack.

usbudc Il driver che gestisce il Controller USB deve agire secondo le specifiche device e fornire verso lo strato software superiore l'interfaccia *Linux usb gadget driver framework*.

gadget driver Driver che determinano le funzionalità esposte dal sistema embedded, compatibili con il device model USB. E' possibile installare più gadget driver alla volta ma in caso di connessione, solo uno può essere attivo.

upper levels I driver gadget possono utilizzare tutte le funzionalità Linux per svolgere i loro compiti (filesystems, media capture ...).

USB OTG. Il layer software di gestione del Controller USB, nel caso questo disponga della funzionalità OTG e la si volesse sfruttare appieno, deve poter definire per ogni connessione il ruolo del Controller nella comunicazione ed attivare lo stack software adatto. I driver per controller OTG fanno parte della famiglia dei gadget driver.

Codice di supporto per i Controller USB degli Application Processor i.MX

Daniel Mack <daniel@caiaq.de> , basandosi su un concept iniziale di Sascha Hauer, recentemente ha lavorato per ottenere in kernel il supporto in modalità Host dei controller USB della famiglia di Application Processor i.MX consiste in:

- Ampliamento dell'interfaccia Linux OTG (*/include/linux/usb/otg.h*) per definire le operazioni fondamentali di un modulo transceiver.
- Supporto a transceiver OTG controllati tramite interfaccia ULPI con un driver apposito inserito nella directory */drivers/usb/otg/*.
- Supporto alle operazioni fondamentali delle porte ULPI con codice di piattaforma inserito in */arch/arm/plat-mxc/*.
- Sviluppo del codice specifico di gestione dei Controller USB per il driver *ehci-hcd.*
- Inizializzazione e registrazione dei driver nel codice di supporto della board (presentata per la board lilly1131).

La serie di patch più aggiornata è stata presentata nelle mailing list *linux-arm-kernel* e *linux-usb* con il titolo "`Patches for Freescale MXC SoCs and direct ULPI communication [v3]`".

Applicata ed adattata al repository locale del kernel (portando il codice di inizializzazione nel file *armadillo5x0.c* con le adeguate modifiche) è possibile abilitare i due controller USBOTG e USBH2 in modalità Host con piene funzionalità.

Modifiche al codice di supporto. Di seguito sono riportate le modifiche al file *armadillo5x0.c*.

Listing 6.27: armadillo5x0-08ToI2C.patch

```
diff --git a/arch/arm/mach-mx3/Kconfig b/arch/arm/mach-mx3/Kconfig
index 851f245..8cf0e9c 100644
--- a/arch/arm/mach-mx3/Kconfig
+++ b/arch/arm/mach-mx3/Kconfig
@@ -89,6 +89,7 @@ config MACH_PCM043
 config MACH_ARMADILLO5X0
        bool "Support Atmark Armadillo-500 Development Base Board"
        select ARCH_MX31
+       select MXC_ULPI
        help
```

```
          Include support for Atmark Armadillo-500 platform. This includes
          specific configurations for the board and its peripherals.
diff --git a/arch/arm/mach-mx3/armadillo5x0.c b/arch/arm/mach-mx3/armadillo5x0.c
index d10ee27..936fc86 100644
--- a/arch/arm/mach-mx3/armadillo5x0.c
+++ b/arch/arm/mach-mx3/armadillo5x0.c
@@ -36,6 +36,9 @@
 #include <linux/input.h>
 #include <linux/gpio_keys.h>
 #include <linux/i2c.h>
+#include <linux/delay.h>
+#include <linux/usb/otg.h>
+#include <linux/usb/ulpi.h>

 #include <mach/hardware.h>
 #include <asm/mach-types.h>
@@ -52,6 +55,8 @@
 #include <mach/ipu.h>
 #include <mach/mx3fb.h>
 #include <mach/mxc_nand.h>
+#include <mach/mxc_ehci.h>
+#include <mach/ulpi.h>

 #include "devices.h"
 #include "crm_regs.h"
@@ -105,6 +110,127 @@ static int armadillo5x0_pins[] = {
        MX31_PIN_CSPI2_MISO__SDA,
 };

+/* USB */
+#define OTG_RESET IOMUX_TO_GPIO(MX31_PIN_STXD4)
+#define USBH2_RESET IOMUX_TO_GPIO(MX31_PIN_SCK6)
+#define USBH2_CS IOMUX_TO_GPIO(MX31_PIN_GPIO1_3)
+
+#define USB_PAD_CFG (PAD_CTL_DRV_MAX | PAD_CTL_SRE_FAST | PAD_CTL_HYS_CMOS | \
+                     PAD_CTL_ODE_CMOS | PAD_CTL_100K_PU)
+
+static int usbotg_init(struct platform_device *pdev)
+{
+       unsigned int pins[] = {
+               MX31_PIN_USBOTG_DATA0__USBOTG_DATA0,
+               MX31_PIN_USBOTG_DATA1__USBOTG_DATA1,
+               MX31_PIN_USBOTG_DATA2__USBOTG_DATA2,
+               MX31_PIN_USBOTG_DATA3__USBOTG_DATA3,
+               MX31_PIN_USBOTG_DATA4__USBOTG_DATA4,
+               MX31_PIN_USBOTG_DATA5__USBOTG_DATA5,
+               MX31_PIN_USBOTG_DATA6__USBOTG_DATA6,
+               MX31_PIN_USBOTG_DATA7__USBOTG_DATA7,
+               MX31_PIN_USBOTG_CLK__USBOTG_CLK,
+               MX31_PIN_USBOTG_DIR__USBOTG_DIR,
+               MX31_PIN_USBOTG_NXT__USBOTG_NXT,
+               MX31_PIN_USBOTG_STP__USBOTG_STP,
+       };
+
+       mxc_iomux_setup_multiple_pins(pins, ARRAY_SIZE(pins), "USB_OTG");
+
```

```
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_DATA0, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_DATA1, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_DATA2, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_DATA3, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_DATA4, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_DATA5, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_DATA6, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_DATA7, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_CLK, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_DIR, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_NXT, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBOTG_STP, USB_PAD_CFG);
+
+       /* Chip already enabled by hardware */
+       /* OTG phy reset*/
+       gpio_request(OTG_RESET, "USB-OTG-RESET");
+
+       if (gpio_direction_output(OTG_RESET, 1/*HIGH*/)) {
+               pr_warning("Failed to reset usbotg phy \n");
+               return -1;
+       }
+        gpio_set_value(OTG_RESET, 0/*LOW*/);
+        msleep(5);
+        gpio_set_value(OTG_RESET, 1/*HIGH*/);
+
+       return 0;
+}
+
+static int usbh2_init(struct platform_device *pdev)
+{
+       int pins[] = {
+               MX31_PIN_USBH2_DATA0__USBH2_DATA0,
+               MX31_PIN_USBH2_DATA1__USBH2_DATA1,
+               IOMUX_MODE(MX31_PIN_STXD3, IOMUX_CONFIG_FUNC),
+               IOMUX_MODE(MX31_PIN_SRXD3, IOMUX_CONFIG_FUNC),
+               IOMUX_MODE(MX31_PIN_SCK3, IOMUX_CONFIG_FUNC),
+               IOMUX_MODE(MX31_PIN_SFS3, IOMUX_CONFIG_FUNC),
+               IOMUX_MODE(MX31_PIN_STXD6, IOMUX_CONFIG_FUNC),
+               IOMUX_MODE(MX31_PIN_SRXD6, IOMUX_CONFIG_FUNC),
+               MX31_PIN_USBH2_CLK__USBH2_CLK,
+               MX31_PIN_USBH2_DIR__USBH2_DIR,
+               MX31_PIN_USBH2_NXT__USBH2_NXT,
+               MX31_PIN_USBH2_STP__USBH2_STP,
+       };
+
+       mxc_iomux_setup_multiple_pins(pins, ARRAY_SIZE(pins), "USB_H2");
+
+       mxc_iomux_set_pad(MX31_PIN_USBH2_CLK, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBH2_DIR, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBH2_NXT, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBH2_STP, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBH2_DATA0, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_USBH2_DATA1, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_SRXD6, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_STXD6, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_SFS3, USB_PAD_CFG);
```

```
+       mxc_iomux_set_pad(MX31_PIN_SCK3, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_SRXD3, USB_PAD_CFG);
+       mxc_iomux_set_pad(MX31_PIN_STXD3, USB_PAD_CFG);
+
+       mxc_iomux_set_gpr(MUX_PGP_UH2, true);
+
+
+       /* Enable the chip */
+       gpio_request(USBH2_CS, "USB-H2-CS");
+       gpio_direction_output(USBH2_CS, 0/*Enabled*/);
+
+       /* H2 phy reset*/
+       gpio_request(USBH2_RESET, "USB-H2-RESET");
+
+       if (gpio_direction_output(USBH2_RESET, 1/*HIGH*/)) {
+               pr_warning("Failed to reset usbh2 phy \n");
+               return -1;
+       }
+        gpio_set_value(USBH2_RESET, 0/*LOW*/);
+        msleep(5);
+        gpio_set_value(USBH2_RESET, 1/*HIGH*/);
+
+       return 0;
+}
+
+static struct mxc_usbh_platform_data usbotg_pdata = {
+       .init   = usbotg_init,
+       .portsc = MXC_EHCI_MODE_ULPI | MXC_EHCI_UTMI_8BIT,
+       .flags  = MXC_EHCI_POWER_PINS_ENABLED,
+};
+
+static struct mxc_usbh_platform_data usbh2_pdata = {
+       .init   = usbh2_init,
+       .portsc = MXC_EHCI_MODE_ULPI | MXC_EHCI_UTMI_8BIT,
+       .flags  = MXC_EHCI_POWER_PINS_ENABLED,
+};
+
+
 /* RTC over I2C*/
 #define ARMADILLO5X0_RTC_GPIO  IOMUX_TO_GPIO(MX31_PIN_SRXD4)

@@ -393,6 +519,15 @@ static void __init armadillo5x0_init(void)
        if (armadillo5x0_i2c_rtc.irq == 0)
                pr_warning("armadillo5x0_init: failed to get RTC IRQ\n");
        i2c_register_board_info(1, &armadillo5x0_i2c_rtc, 1);
+
+       /* USB */
+       usbotg_pdata.otg = otg_ulpi_create(&mxc_ulpi_access_ops,
+                                  USB_OTG_DRV_VBUS | USB_OTG_DRV_VBUS_EXT);
+       usbh2_pdata.otg = otg_ulpi_create(&mxc_ulpi_access_ops,
+                                  USB_OTG_DRV_VBUS | USB_OTG_DRV_VBUS_EXT);
+
+       mxc_register_device(&mxc_otg_host, &usbotg_pdata);
+       mxc_register_device(&mxc_usbh2, &usbh2_pdata);
 }
```

```
 static void __init armadillo5x0_timer_init(void)
__
1.5.4.3
```

A differenza della patch di Daniel non è stato inizializzato il controller USBH1, è stato introdotto un inpulso di reset ai transceiver nelle rispettive funzioni di inizializzazione e sono state rimosse le modifiche al registro di controllo del modulo multiplexer GPR per l'interfaccia relativa al controller USBOTG.

Capitolo 7

Android su piattaforma Freescale i.MX31L

Nel capitolo precedente si sono descritte le fasi che hanno portato alla creazione della piattaforma di supporto della board Atmark Armadillo 500 per il kernel Linux.

Non tutte le periferiche sono state abilitate ma il livello di multimedialità, di supporto ai dispositivi di I/O e di memorizzazione di massa sono sufficienti per eseguire una verifica del Sistema Operativo Android nella board Armadillo 500.

Al momento della redazione di questo documento il repository Android risiede approssimativamente alla versione 1.5 (cupcake).

7.1 Il codice sorgente di Android

Seguendo la traccia pubblicata nel sito ufficiale del progetto open source[1] si procede con il download dell'utility `repo` utilizzata per le operazioni di base di gestione del repository locale:

7.1.1 Installare repo

Android utilizza Git come software di version management per ogni componente del progetto: ogni modulo Android possiede un proprio repository git autonomo. `repo` è uno script capace di automatizzare le opera-

[1] `http://source.android.com/download`

zioni che altrimenti dovrebbero essere eseguite in serie per ogni singolo repository unificandole in un singolo comando globale.

```
$ curl http://android.git.kernel.org/repo >~/bin/repo
$ chmod a+x ~/bin/repo
```

Si faccia riferimento al documento [39] per una guida completa dei comandi dello script.

7.1.2 Download dei sorgenti

Si procede ora definendo la directory nel quale sarà creato il repository ed all'inizializzazione dello stesso:

```
$ mkdir $PRJROOT/android-sources
$ cd $PRJROOT/android-sources
$ repo init -u git://android.git.kernel.org/platform/manifest.
   git
```

Se si vuole eseguire il *checkout* di un branch diverso da `master` deve essere specificato il parametro `-b branchName`, ad esempio:

```
$ repo init -u git://android.git.kernel.org/platform/manifest.
   git -b cupcake
```

Al termine delle operazioni di inizializzazione del repository, verrà richiesto di inserire il nome dello sviluppatore ed un indirizzo *gmail* valido che verranno utilizzati nel caso si volessero sottoporre a review e pubblicare le eventuali modifiche apportate al codice sorgente [38].

Sincronizzazione del repository locale con la versione remota. Per sincronizzare il repository locale con le ultime modifiche remote è sufficiente eseguire:

```
$ cd $PRJROOT/android-sources
$ repo sync
```

7.1.3 Prima build di Android

Per verificare la funzionalità del repository creato si può eseguire una prima build dell'intero stack con:

```
$ cd $PRJROOT/android-sources
$ make
```

Se il processo andrà a buon fine verranno prodotti nella directory *out/target/product/generic/* i file immagine:

ramdisk.img Contiene quei file strettamente necessari per partire con il processo di inizializzazione dell'ambiente Android:

- Lo scheletro del root-filesystem Android.
- Il file binario */init* di inizializzazione dello stack.
- Gli script che determinano le operazioni eseguite dal processo `init` : */init.rc* e */init.machineName.rc*.

system.img Contiene tutti i file binari e di configurazione dello stack Android.
Dovrà essere montato nella directory */system*.

userdata.img Contiene una versione iniziale della partizione dedicata ai dati utente.
Dovrà essere montato nella directory */data*.

La toolchain utilizzata è compilata per architettura ARMv5te e risiede nella directory *prebuilt/linux-x86/toolchain/arm-eabi-4.2.1/bin* del repository.

7.2 Ottenere un kernel valido

La directory *kernel* del repository Android contiene il repository git locale del kernel Android Linux-derived. Le corrispondenti versioni del kernel Linux supportatate sono[2]:

```
$ cd $PRJROOT/android-sources/kernel
$ git branch -r
  korg/android-2.6.25
  korg/android-2.6.27
  korg/android-2.6.29
  korg/android-goldfish-2.6.27
  korg/android-goldfish-2.6.29
  m/master
```

[2]l'opzione `-r` indica al comando git `branch` di visualizzare non solo i branch locali del progetto ma anche quelli reomoti.

Quale versione del kernel è quella favorevole? Dato che la piattaforma di supporto alla board Armadillo 500 è entrata a far parte del mainline kernel dalla versione maggiore 2.6.31 e che le versioni precedenti del kernel contengono errori significativi che ne precludono l'utilizzo (i bug-fix dei driver Seriale e NAND sono necessari, inoltre il sottosistema USB si basa esplicitamente sulla versione della piattaforma i.MX MXC contenuta nel kernel Linix 2.6.31), deve essere eseguita una fusione consistente tra le due versioni del kernel (Android e Linux mainline) a partire da una versione favorevole.

L'operazione di avanzamento del kernel Android `android-2.6.29` allo stato mainline non è conveniente: la mole di patch che dovrebbero essere applicate e le inconsistenze da ripianare (il merge dei differenti repository di sviluppo nel mainline kernel richiede spesso la risoluzione di inconsistenze da risolvere a mano) sono tali da rendere il processo inutilmente faticoso.

Neppure è possibile ora applicare l'intera sequenza di patch Android al mainline kernel Linux (ad ora alla versione `2.6.32-rc1`) dato che il codice di Power Management Linux ha subito un'evoluzione tale da non rendere applicabili le relative modifiche Android.

La soluzione proposta è quella di utilizzare una versione intermedia del kernel: la `2.6.30` dove le patch Android sono in maggior parte valide e l'avanzamento della piattaforma per Application Processor della famiglia i.MX è in maggior parte lineare.

Per inizializzare un nuovo repository locale del kernel Linux alla versione selta si esegue:

```
$ mkdir $PRJROOT/kernelAndroidArmadillo500
$ cd $PRJROOT/kernelAndroidArmadillo500
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/
   torvalds/linux-2.6.git
$ cd linux-2.6
$ git checkout v2.6.30
```

7.2.1 Le patch Android

Ottenere le patch

In riferimento all'articolo [29] per ottenere la serie di patch Android da applicare al kernel `2.6.30` si procede come segue.

Il repository locale del kernel Android deve essere ricollocato nel branch `android-2.6.29` (dato che ad ora risiede per default nel branch `android-2.6.27`):

```
$ cd $PRJROOT/android-sources/kernel
$ git checkout korg/android-2.6.29
```

La serie di patch è ottenuta con il comando:

```
$ git format-patch v2.6.30 -o androidPatches
```

`format-patch` provvederà a creare nella directory `androidPatches` la serie completa di patch che dovrebbe dividere il kernel `2.6.30` dal kernel `android-2.6.29`.

Questa operazione è possibile dato che attraverso git, il repository del kernel Android tiene traccia dell'avanzamento di versione del kernel mainline. Ciò comporta che, se alcune commit eseguite nel branch Android, sottoposte a review, vengono accettate ed applicate anche nella versione mainline del kernel (fino alla versione indicata), allora le relative patch non dovrebbero far parte del set prodotto dal comando.

Nel caso del branch Android del kernel Linux, per motivi non compresi, non è vera questa proprietà. Quindi, dalla serie di patch prodotta, dovranno essere individuate ed eliminate le patch già presenti nel kernel `2.6.30`.

Applicare le patch Android al kernel 2.6.30

Recuperate le patch Android ottenute come sopra con:

```
$ cd $PRJROOT/
$ cp android-sources/kernel/androidPatches
   kernelAndroidArmadillo500/linux-2.6 -r
$ cd kernelAndroidArmadillo500/linux-2.6
```

Si procede nella sequenza di applicazione.

Individuare le patch già applicate. Innanzitutto dall'insieme di patch prodotte devono essere eliminate quelle già presenti nel kernel Linux `2.6.30`. A questo proposito può venire in aiuto un semplice script:

Listing 7.1: testPatches.sh

```
#!/bin/bash
```

```
# Utilizzo: testPatches.sh patchDir pruneDir
# Deve essere eseguito dalla root directory di un repository git

# -h sopprime l'informazione␣"nomeFile:"␣all'inizio della riga risultato
grep -h  "Subject:" $1/*.patch > $1/subjects.txt

# Elimino la porzione dell'oggetto che non andrà a formare la stringa
# di commit.
sed "s/Subject: \[PATCH\] \[ARM\] //" $1/subjects.txt > $1/subj1.txt
sed "s/Subject: \[PATCH\] //" $1/subj1.txt > $1/subj.txt

# Ricerca di ogni singola patch nelle stringhe di commit riguardanti
# il sottoalbero di prune indicato.
count=0
while read line ;do
        git log $2 | grep "$line"
        count=$[$count + 1]
        echo $count
done < $1/subj.txt

# Rimuovo i file temporanei
rm -f $1/subj.txt $1/subj1.txt $1/subjects.txt
```

Salvato nella directory `~/bin` e configurato per l'esecuzione `$ chmod +x ~/bin/testPatches.sh` può essere utilizzato come segue:

```
$ testPatches.sh androidPatches
```

Le patch relative alle stringhe *Subject:* stampate, devono essere eliminate dalla cartella `androidPatches`.

Applicare le patch. Innanzitutto è bene creare un nuovo branch locale, significativo per l'obbiettivo del presente repository:

```
$ git branch androidArmadillo500
```

La sequenza di patch viene applicata con il comando:

```
$ cat androidPatches/* | git am -k -3
```

`git am` applica ed esegue il commit di ogni singola "mail" ricevuta nello standard-in. I file prodotti da `git format-patch` sono proprio formattati come mail con campi *From:, Subject:, Date:, Signed-off:* dai quali `git am` preleva i dati per eseguire la relativa commit locale.

Le opzioni `-k` e `-3` indicano rispettivamente a `git am` di non tentare di ripulire la stringa *Subject:* da porzioni inutili ("Re:", "[PATCH]" ..) e tentare il metodo *three way merge* nel caso di conflitto nell'applicazione della patch corrente.

`git am` crea un database delle mail-patch a lui inviate nella directory *.dotest*. Nel caso si verificassero errori di applicazione che richiedono l'intervento a mano, `git am` termina temporaneamente l'applicazione della sequenza lasciando il marge della patch incriminata allo sviluppatore.

Per riprendere nella sequenza di applicazione dalla patch incriminata `git am –resolved`, mentre con `git am –skip` la sequenza di applicazione continuerà saltando la patch corrente.

Nel caso si volesse annullare la sequenza di applicazione corrente e ritornare allo stato iniziale:

```
$ rm -r .dotest
$ git reset --hard androidArmadillo500
```

Patch Tralasciate. Non tutte le patch risultanti sono state applicate per problemi di incompatibilità con la versione scelta del kernel. Si trattano per lo più di bug-fix proposti dal gruppo Android ad aree del kernel non strettamente legate allo stack Android non accettati finora dalla comunità Linux.

Tra le patch non applicate c'è il supporto in kernel del filesystem yaffs2. Per problemi nell'applicazione e dato che questo filesystem non verrà utilizzato nei test successivi, si è deciso di non implementare tale funzionalità.

Patch Applicate. Le patch applicate introducono nel kernel tutte le funzionalità necessarie allo stack Android assenti nella versione `2.6.30` del kernel Linux quali: *ashmem*, *pmem*, *wake_lock*, *gadget driver per adb*, *alarm*, *Network Security*.

Il modulo *ashmem* è stato sviluppato per kernel `2.6.27` e necessita di una piccola modifica per essere applicato nella versione del kernel Linux scelta. La funzione:

mm/shmem.c:**`int shmem_zero_setup(struct vm_area_struct *vma)`**

Differisce nelle due versioni del kernel Linux (`2.6.27` e `2.6.30`) nella presenza della chiamata al metodo **`ima_shm_check(file)`** parte dell'architettura di sicurezza *Integrity Measurement Architecture(IMA)*. Perché anche le pagine allocate da *ashmem* vengano considerate da questa architettura, allora la chiamata a **`ima_shm_check(file)`** è stata inserita

nella nuova funzione introdotta dalla patch, punto di collegamento tra il modulo Anonymous SHared MEMory system e Linux shmem:

mm/shmem.c:**void shmem_set_file(struct vm_area_struct *vma, struct file *file)**

In allegato a questo documento sarà presente una cartella *androidPatches* dove all'interno il file *0001-Android-2.6.29-partial-working-merge.patch* unisce in una unica patch tutte quelle applicate (presenti singolarmente nella sottodirectory *applicate*) mentre quelle non applicate risiederanno nella sottodirectory *NonApplicate*.

E' utile a questo punto segnare il livello raggiunto con un tag git al repository:

```
$ git tag android-done
```

Versione del repository alla quale è possibile ritornare con il comando:

```
$ git reset --hard android-done
```

7.2.2 Avanzare la piattaforma i.MX

Ora l'obbiettivo è quello di aggiornare le sezioni del kernel che formano il codice di supporto per tutte le board che montano processori della famiglia i.MX.

Tali sezioni del kernel risiedono principalmente nelle directory *arch/arm/plat-mxc* e *arch/arm/mach-mx3*, alle quali vanno a sommarsi i driver di periferica posti in modo consistente all'interno del kernel tree.

Ottenere la serie di patch corretta

Per ottenere la serie di patch da applicare al branch locale `androidArmadillo500` creato, è utile installare l'utility grafica `Qgit`:

```
$ apt get --install qgit
```

Se si esegue `qgit` all'interno di un repository Git, questa analizzerà le informazioni contenute nel relativo database index e permetterà di manipolare graficamente il repository.

Le funzioni "*Filter by tree*" e "*Format patch..*" sono utili allo scopo: abilitando la visualizzazione laterale dell'albero delle directory è possibile

selezionare uno o più elementi e cliccando sull'icona "Filter by tree" verranno filtrate le patch visualizzate nella finestra centrale alle sole che nelle loro modifiche toccano gli elementi selezionati.

Selezionando poi un range di patch è possibile esportarle con "*Format patch..*" in una directory selezionata.

Allora le operazioni da eseguire per ottenere il set di patch di avanzamento sono:

1. Nel repository locale `kernelAndroidArmadillo500/linux-2.6` verificare le ultime commit alle due directory *`arch/arm/plat-mxc`* e *`arch/arm/mach-mx3`* con:

```
$ cd $PRJROOT/kernelAndroidArmadillo500/linux-2.6
$ git log arch/arm/plat-mxc
commit 7b9020badf78327b3fcb567b466a1dd4d33710ce
Author: Uwe Kleine-König <u.kleine-koenig@pengutronix.de>
Date:   Tue Apr 21 22:56:14 2009 +0200

    mx27ads: move PBC mapping out of vmalloc space
...

$ git log arch/arm/mach-mx3
commit 6b4bfb87b638a4f114dfb6f72f4ac1be88a4ebe4
Author: Rabin Vincent <rabin@rab.in>
Date:   Tue May 26 22:31:46 2009 +0530

    mx[23]: don't put clock lookups in __initdata
...
```

2. Eseguire `qgit` nella directory `$PRJROOT/kernelLinus/linux-2.6`, selezionare come elementi di filtro le due directory *`arch/arm/plat-mxc`* e *`arch/arm/mach-mx3`* ed applicare "*Filter by tree*".

3. Selezionare tutte le patch, dalla più recente, alla più vecchia di quelle verificate nel repository, `kernelAndroidArmadillo500` ed esportarle con "*Format patch..*" nella directory `kernelAndroidArmadillo500/linux-2.6/platformPatches`.

Ciò che distingue questa sequenza di azioni dai comandi git:

```
$ cd $PRJROOT/kernelLinus/linux-2.6
$ git format-patch 6b4bfb87b638a4f114dfb6f72f4ac1be88a4ebe4 arch
   /arm/mach-mx3
```

```
e
$ git format-patch 7b9020badf78327b3fcb567b466a1dd4d33710ce arch
    /arm/plat-mxc
```

è la consistenza delle patch create: in questo ultimo modo `format-patch` filtra non solo nel set di patch quelle che intervengono nelle due directory ma anche le patch stesse, eliminando gli interventi congiunti in altre porzioni del kernel rendendo così il risultato inconsistente.

Applicare il set di patch ottenuto

Dopo un'analisi a mano del set ottenuto in precedenza (Qgit, nel risolvere gli eventi di merge tra due branch avvenuti nel range di patch selezionato, crea oltre alla serie di patch selezionate l'intera serie del branch di merge sovrapponendole) è possibile procedere nell'applicazione delle patch.

In allegato a questo documento sarà presente la directory *platformPatches* contenente la serie consistente.

Per applicarla si esegua lo script seguente, da salvare nella directory *kernelAndroidArmadillo500/linux-2.6* dove è stata precedentemente copiata *platformPatches*.

Listing 7.2: updatePlatform.sh

```
#!/bin/bash

git reset --hard android-done

cat platformPatches/*.patch | git am -k -3
cp platformPatches/Kconfig arch/arm/mach-mx3/
git add arch/arm/mach-mx3/Kconfig
git commit -m 'Kconfig merge'
cat platformPatches/armadilloSpec/* | git am -k -3
```

A questo punto il kernel contenuto nel repository locale *kernelAndroidArmadillo500/linux-2.6* è pronto per poter essere configurato e compilato.

7.2.3 Configurare il kernel Android ottenuto.

Per configurare il kernel Android ottenuto si esegue:

```
$ cd $PRJROOT/kernelAndroidArmadillo500/linux-2.6
$ make menuconfig
```

Dopo aver selezionato tutte le opzioni riguardanti la board Armadillo500 (si veda la sequenza di porting nella sezione 6.3 di questo documento) si procede ad abilitare le funzionalità necessarie ad Android:

ashmem In `General setup --> [*] Enable the Anonymous Shared Memory Subsystem`.

wake_lock In `Power management options --> [*] Power Management support , [*] Wake lock, [*] Userspace wake locks, [*] Early suspend`.

security In `[*] Networking support --> Networking options -->, [*] Only allow certain groups to create sockets`.

binder, log, ram buffer, low memory killer In `Device Drivers --> [*] Staging drivers --> Android --> [*] Android Drivers, [*] Android Binder IPC Driver, <*> Android log driver, [*] Android RAM buffer console, [*] Android Low Memory Killer`.

Infine deve essere abilitato il sottosistema *Event interface* `Device Drivers --> Input device support --> <*> Event interface` per compatibilità con il sistema di Input Android.

Così configurato il kernel è pronto per la compilazione ed il trasferimento nella board.

7.3 Personalizzare il processo di build per la board Armadillo 500

Gli script di generazione dello stack risiedono nella directory *`build`* del repository Android dove è possibile trovare inoltre il documento *Android Build System*[2] dogumentazione parziale ma utile degli stessi.

Il sistema di build di Android permette la definizione di *prodotti target* differenti: definendo un prodotto targhet è possibile indicare quali moduli Android generare ed installare (librerie ed applicazioni), personalizzare proprietà di sistema ed indicare una lista di file da inserire staticamente nel filesystem generato per tale prodotto.

Prodotti nativi. Nella directory *build/target/product* sono definiti i *prodotti targhet* utilizzabili nativamente, che sono:

min_dev: Prodotto minimale. Genera uno stack Android di test di funzionalità per un dispositivo includendo, oltre alle librerie di sistema, solamente i moduli strettamente necessari all'avvio e navigazione nell'ambiente Android.

generic: Genera un prodotto contenente tutti i moduli e applicazioni di uso comune.

sim e sdk: Prodotti generici pensati per supportare l'emulatore contenuto nell'SDK.

7.3.1 Definire un prodotto.

Un prodotto è definito attraverso il suo file *nome_prodotto.mk* nella directory *build/target/product/*.

Le variabili importanti che determinano un prodotto sono:

PRODUCT_NAME, PRODUCT_DEVICE, PRODUCT_BRAND, PRODUCT_MODEL,

PRODUCT_MANUFACTURER: Identificano il prodotto, solo le prime tre sono obbligatorie ed in particolare PRODUCT_NAME viene utilizzata per la ricerca degli script specifici di generazione della piattaforma.

PRODUCT_LOCALES: Lista delle stringhe di localizzazione da installare nel dispositivo. (*es: it_IT en_GB* ..)

PRODUCT_PACKAGES: Lista dei moduli da inatallare nel dispositivo.

PRODUCT_PROPERTY_OVERRIDES: Lista di elementi "*nome_proprietà=valore*" per impostare determinate proprietà di sistema.

PRODUCT_COPY_FILES: Lista di elementi "*path_host:path_targhet*" dove è possibile definire delle installazioni statiche di determinati files nella piattaforma. *path_host* è relativo alla root directory del repository Android e *path_targhet* è relativo alla root directory del root-fylesystem targhet.

PRODUCT_OTA_PUBLIC_KEYS: Lista di chiavi di cifratura valide per verificare le applicazioni certificate.

PRODUCT_POLICY: Definisce quale set di regole di sicurezza utilzzare (è stata utilizzata la sola *android.policy_phone*).

La definizione di prodotto può essere effettuata sulla base di un prodotto parde ereditandone le impostazioni. Questa operazione viene effettuata con la stringa:

```
$(call inherit-product, $(SRC_TARGET_DIR)/product/parentProduct.
    mk)
```

Che si occupa, oltre che a definire le variabili di identificazione, di aggiungere alle liste del prodotto padre i contenuti definiti per il prodotto presente.

Successivamente a questa chiamata, dovranno essere sovrascritte le variabili di identificazione del nuovo prodotto.

Collegare un nuovo prodotto al processo di build. Perché un nuovo file di definizione di prodotto venga considerato nel processo di build, deve essere inserito nella lista `PRODUCT_MAKEFILES` contenuta nel file *build/target/product/AndroidProducts.mk*.

7.3.2 Impostzioni board-specific di generazione dello stack.

La definizione di prodotto presentata fino ad ora, indica al processo di generazione i contenuti e le proprietà dello stack Android voluto ma non entrano nel merito tecnico di collegamento verso l'hardware: kernel, tipologia di filesystem etc.

Ad un prodotto target sono associati determinati script di extra-configurazione e generazione della piattaforma, questi sono definiti nei file *BoardConfig.mk* e *AndroidBoard.mk* contenuti nella directory *build/target/board/product_name/*.

***BoardConfig.mk*:** Viene analizzato nella fase di configurazione del processo di build e definisce alcune variabili d'ambiente che influiscono nella generazione dei moduli :

`TARGET_BOARD_PLATFORM`: Definisce il nome della piattaforma, verrà impostata la proprietà di sistema *ro.board.platform=$TARGET_BOARD_PLATFORM*.

`TARGET_NO_BOOTLOADER`: Se *true* non viene generato il bootloader.

`TARGET_NO_KERNEL`: Se *true* non viene generato il kernel.

`TARGET_NO_RADIOIMAGE`: Se *true* non vengono generate le librerie di gestione delle comunicazioni radio.

`HAVE_HTC_AUDIO_DRIVER`: Se *true* viene generato lo stub-driver Android in grado di gestire l'interfaccia audio attraverso il driver specifico HTC.

`USE_CAMERA_STUB`: Dato che l'assenza di un driver-stub di gestione del sensore video pregiudica il funzionamento del Media Server, se *true* genera un driver-stub dummy che non fa nulla.

`TARGET_USERIMAGES_USE_EXT2`: Se *true* allora i file immagine Android utilizzeranno il filesystem *ext2*, *yaffs2* altrimenti.

***AndroidBoard.mk*:** Parte integrante della fase esecutiva di build, è utilizzato per svolgere dei compiti specifici board-related come la generazione della tabella dei caratteri.

La patch risultante. Ecco le modifiche introdotte nell'architettura di generazione dello stack Android:

Listing 7.3: Add-Armadillo500-board-product-definition.patch

```
From e66b631aecc95eee3ac815c1dd5d2a79401620b0 Mon Sep 17 00:00:00 2001
From: Alberto Panizzo <maramaopercheseimorto@gmail.com>
Date: Fri, 16 Oct 2009 22:03:44 +0200
Subject: [PATCH] Add Armadillo500 board product definition.

---
 target/board/armadillo500/AndroidBoard.mk          |    6 ++++++
 target/board/armadillo500/BoardConfig.mk           |   16 ++++++++++++++++
 target/board/armadillo500/armadillo_keylayout.kl   |    2 ++
 target/product/AndroidProducts.mk                  |    3 ++-
 target/product/armadillo500.mk                     |    7 +++++++
 5 files changed, 33 insertions(+), 1 deletions(-)
 create mode 100644 target/board/armadillo500/AndroidBoard.mk
 create mode 100644 target/board/armadillo500/BoardConfig.mk
 create mode 100644 target/board/armadillo500/armadillo_keylayout.kl
 create mode 100644 target/product/armadillo500.mk

diff --git a/target/board/armadillo500/AndroidBoard.mk b/target/board/
    armadillo500/AndroidBoard.mk
new file mode 100644
index 0000000..3358e6a
--- /dev/null
+++ b/target/board/armadillo500/AndroidBoard.mk
@@ -0,0 +1,6 @@
+LOCAL_PATH := $(call my-dir)
+
+file := $(TARGET_OUT_KEYLAYOUT)/armadillo_keylayout.kl
+ALL_PREBUILT += $(file)
+$(file) : $(LOCAL_PATH)/armadillo_keylayout.kl | $(ACP)
```

```
+       $(transform-prebuilt-to-target)
diff --git a/target/board/armadillo500/BoardConfig.mk b/target/board/
    armadillo500/BoardConfig.mk
new file mode 100644
index 0000000..05192ce
--- /dev/null
+++ b/target/board/armadillo500/BoardConfig.mk
@@ -0,0 +1,16 @@
+# config.mk
+#
+# Product-specific compile-time definitions.
+#
+
+TARGET_BOARD_PLATFORM := armadillo500
+TARGET_NO_BOOTLOADER := true
+TARGET_NO_KERNEL := true
+TARGET_NO_RADIOIMAGE := true
+HAVE_HTC_AUDIO_DRIVER := false
+BOARD_USES_GENERIC_AUDIO := false
+
+
+#Use fake camera for now
+USE_CAMERA_STUB := true
+TARGET_USERIMAGES_USE_EXT2 := true
diff --git a/target/board/armadillo500/armadillo_keylayout.kl b/target/board/
    armadillo500/armadillo_keylayout.kl
new file mode 100644
index 0000000..5a3592d
--- /dev/null
+++ b/target/board/armadillo500/armadillo_keylayout.kl
@@ -0,0 +1,2 @@
+key 28   MENU          WAKE
+key 158   BACK          WAKE
diff --git a/target/product/AndroidProducts.mk b/target/product/AndroidProducts.
    mk
index 1bf3c3f..37c14dc 100644
--- a/target/product/AndroidProducts.mk
+++ b/target/product/AndroidProducts.mk
@@ -30,4 +30,5 @@ PRODUCT_MAKEFILES := \
     $(LOCAL_DIR)/min_dev.mk \
     $(LOCAL_DIR)/sdk.mk \
     $(LOCAL_DIR)/sim.mk \
-    $(LOCAL_DIR)/generic_with_google.mk
+    $(LOCAL_DIR)/generic_with_google.mk \
+    $(LOCAL_DIR)/armadillo500.mk
diff --git a/target/product/armadillo500.mk b/target/product/armadillo500.mk
new file mode 100644
index 0000000..1e061b8
--- /dev/null
+++ b/target/product/armadillo500.mk
@@ -0,0 +1,7 @@
+$(call inherit-product, $(SRC_TARGET_DIR)/product/min_dev.mk)
+
+# Overrides
+PRODUCT_BRAND := generic
+PRODUCT_NAME := armadillo500
```

```
+PRODUCT_DEVICE := armadillo500
+PRODUCT_LOCALES := en_US
—
1.5.4.3
```

7.3.3 Modificatori di prodotto

Il sistema di build definisce dei *modificatori di prodotto* che determinano quale versione del prodotto si vuole generare.

Ogni componente Android (libreria o applicazione) può esporre un tag che ne modifica la presenza o meno nella versione del prodotto scelta attraverso la lista `LOCAL_MODULE_TAGS`. I tag nativi sono: *eng*, *debug*, *user* e *development*, e permettono le versioni di prodotto date dai seguenti modificatori:

eng Modificatore di default, non valuta il tag dei moduli per la loro inclusione nello stack genrato.
I comandi `make` e `make eng` producono lo stesso effetto. *droid* è un alias di *eng*.
Vengono impostate le seguenti variabili di default:
`ro.secure=0` : Le applicazioni non certificate possono eseguire nello stack.
`ro.debuggable=1` : E' generato il server *adb* e viene attivato di default.

user Con `make user` viene generata la versione "finale" dello stack. Vengono installati i soli moduli taggati *user* e quelli non taggati ma inclusi nelle specifiche di prodotto scelto.
Vengono impostate le seguenti variabili di default:
`ro.secure=1` : Le applicazioni non certificate non possono eseguire nello stack.
`ro.debuggable=0` : E' generato il server *adb* ma viene disabilitato di default.

userdebug Con `make userdebug` viene generata una versione simile a *user* contenente inoltre i moduli taggati *debug.*
Vengono impostate le seguenti variabili di default:
`ro.secure=1` : Le applicazioni non certificate non possono eseguire nello stack.
`ro.debuggable=1` : E' generato il server *adb* e viene attivato di default.

Per non utilizzare nella versione scelta moduli compilati con altri modificatori è necessario eseguire

```
$ cd $PRJROOT/android-sources
$ make installclean
```

che provvederà a pulire le directory di output del prodotto Android scelto senza eliminare file binari intermedi all'interno delle directory specifiche dei moduli.

7.3.4 Generare il root-filesystem definito dal prodotto armadillo500

Con le seguenti verranno generati i file immagine appositi per il prodotto armadillo500 secondo le impostazioni definite nei file specifici:

```
$ cd $PRJROOT/android-sources
$ export TARGET_PRODUCT=armadillo500
$ export TARGET_BUILD_TYPE=[debug|release]
$ make
```

I file immagine risiederanno della directory

out/TARGET_BUILD_TYPE==debug?debug:null/target/product/armadillo500

Il file immagine *ramdisk.img* è nel formato *CPIO* compresso mentre *system.img* e *userdata.img* contengono filesystem *ext2*.

Allora è possibile creare il root-fylesystem Android nella directory *rootfsAndroid* con:

```
$ cd $PRJROOT
$ mkdir rootfsAndroid
$ SYSTEMDIR=android-sources/out/debug/target/product/
   armadillo500

$ cp $SYSTEMDIR/ramdisk.img rootfsAndroid/ramdisk.cpio.gz
$ cd rootfsAndroid
$ gzip -d ramdisk.cpio.gz
$ cpio -i -F ramdisk.cpio
$ rm ramdisk.cpio

$ cd $PRJROOT
$ mkdir tmp
$ sudo mount $SYSTEMDIR/system.img tmp -t ext2 -o loop
$ sudo cp tmp/* rootfsAndroid/system -r
```

```
$ sudo umount $PRJROOT/tmp

$ sudo mount $SYSTEMDIR/userdata.img tmp -t ext2 -o loop
$ sudo cp tmp/* rootfsAndroid/data -r
$ sudo umount $PRJROOT/tmp
```

La directory *$PRJROOT/rootfsAndroid* potrà essere impostata come nfs-root nei parametri di boot del kernel creato nella sezione7.2.

Allegati a questo documento nella cartella *ScriptDiBootAndroid* sono presenti gli script di boot utilizzati nei test effettuati da sovrascrivere e copiare nel root-filesystem Android.

7.4 Problemi e Soluzioni

Lo stack Android generato nella sezione precedente è pronto per il primo boot. Dove, come è logico aspettarsi, Android segnala errori e (non tanto logico) impazzisce nello schermo..

In questa sezione verranno descritti i problemi incontrati e le soluzioni proposte.

7.4.1 Framebuffer

Come descritto nella sezione 2.2.2 il server *SurfaceFlinger* opera una sorta di double buffering nella scrittura dei frame nella memoria video. Se il driver video lo supporta, direttamente nel frame buffer, altrimenti viene mappata una seconda area in memoria di dimensione pari ad un frame normale utilizzata come buffer di disegno prima della copia dei dati nella memoria video.

Il test effettuato per determinare o meno se il driver video può supportare tale double buffering è un semplice controllo sulla dimensione dello schermo virtuale:

Listing 7.4: Test effettuato nel file *frameworks/base/libs/ui/EGLDisplaySurface.cpp* ed effetti.

```
status_t EGLDisplaySurface::mapFrameBuffer() {
...
struct fb_var_screeninfo info;
    if (ioctl(fd, FBIOGET_VSCREENINFO, &info) == -1)
        return -errno;
...
```

```
info.yres_virtual = info.yres * 2;
...
uint32_t flags = PAGE_FLIP;
    if (ioctl(fd, FBIOPUT_VSCREENINFO, &info) == -1) {
        info.yres_virtual = info.yres;
        flags &= ~PAGE_FLIP;
        LOGW("FBIOPUT_VSCREENINFO failed, page flipping not supported");
    }

        if (info.yres_virtual < info.yres * 2) {
        info.yres_virtual = info.yres;
        flags &= ~PAGE_FLIP;
        LOGW("page flipping not supported (yres_virtual=%d, requested=%d)",
                info.yres_virtual, info.yres*2);
    }
...
/* buffer    è il puntatore alla memoria del framebuffer.
 * offscreen è l'array globale contenente i due puntatori all'inizio delle
 *           due regioni di double buffering.
 */
...
if (flags & PAGE_FLIP) {
        offscreen[1] = (uint8_t*)buffer + finfo.line_length*info.yres;
    } else {
        offscreen[1] = (uint8_t*)malloc(finfo.smem_len);
        if (offscreen[1] == 0) {
            munmap(buffer, finfo.smem_len);
            return NO_MEMORY;
        }
    }
}
```

Il driver video *mx3fb* alloca una memoria video pari a due frame ed espone uno schermo virtuale doppio in altezza rispetto alla dimensione del singolo frame. Il test effettuato quindi viene passato e la modalità PAGE_FLIP viene abilitata.

I problemi vengono nel momento in cui viene attuata la politica di page flipping:

Listing 7.5: Page flipping nel file *frameworks/base/libs/ui/EGLDisplaySurface.cpp*.

```
uint32_t EGLDisplaySurface::swapBuffers()
{

    /* If we can't do the page_flip, just copy the back buffer to the front */
    if (!(mFlags & PAGE_FLIP)) {
        memcpy(mFb[0].data, mFb[1].data, mInfo.xres*mInfo.yres*2);
        return 0;
    }

    // do the actual flip
```

```
    mIndex = 1 - mIndex;
    mInfo.activate = FB_ACTIVATE_VBL;
    mInfo.yoffset = mIndex ? mInfo.yres : 0;
    if (ioctl(egl_native_window_t::fd, FBIOPUT_VSCREENINFO, &mInfo) == -1) {
        LOGE("FBIOPUT_VSCREENINFO failed");
        return 0;
    }
...

}
```

Il metodo utilizzato per comandare il driver video di traslare l'immagine visualizzata dalla parte alta alla parte bassa dello schermo virtuale (o viceversa) utilizza l'I/O control `FBIOPUT_VSCREENINFO` che risulta, nell'architettura framebuffer Linux non solo in un PAN (Traslazione) dell'area visualizzata, ma forza una reinizializzazione delle impostazioni del driver che continuamente libera e rimappa la memoria del framebuffer reinizializzando i canali DMA verso il modulo IPU ed inviando al modulo DAC continui segnali di reset. Il risultato è un tempo per flip notevole, continui flash a schermo e blocco del sistema dopo un certo lasso di tempo.

Soluzione proposta

La soluzione più semplice ma efficace è quella di forzare il fallimento del test di valutazione del driver video sostituendo la riga:

```
if (info.yres_virtual < info.yres * 2) {
```

con:

```
if (true) {
```

Il sistema risultante ha una grafica fluida ed utilizzabile ma così la memoria allocata ecede nelle dimensioni di un frame video quella effettivamente necessaria.

Per risparmiare memoria allocata ed operare in modo più consistente dovrebbe essere sostituito il tipo di I/O control di aggiornamento del display da `FBIOPUT_VSCREENINFO` a `FBIOPAN_DISPLAY`. Quest'ultimo è definito esplicitamente per le sole operazioni di traslazione del frame visualizzato nella memoria video virtuale risultando in un'operazione più leggera per il driver.

Sfortunatamente il tempo per testare questa opzione non è stato sufficiente: la semplice sostituzione di I/O control non produce l'effetto desiderato risultando in un blocco del driver video.

7.4.2 Battery

Le librerie user-space di gestione dell'energia, dato che non è presente una batteria e non è possibile quindi acquisirne le informazioni di carica, reagiscono come se ne esistesse una con un livello di carica troppo basso: iniziando la transizione del sistema in modalità *Power Save* e visualizzando a schermo l'informativa che indica la necessità di collegare il dispositivo alla rete elettrica.

In *frameworks/base/services/jni/com_android_server_BatteryService.cpp* è contenuta l'implementazione del servizio *BatteryService*.

La patch seguente ne modifica il comportamento affinché, nel caso la piattaforma sia stata generata per il prodotto *armadillo500*, la funzione di aggiornamento dello stato della batteria risponda con livelli di carica normali:

Listing 7.6: Livelli di carica della batteria ottimali per la piattaforma *armadillo500*.

```
diff --git a/services/jni/com_android_server_BatteryService.cpp b/services/jni/
    com_android_server_BatteryService.cpp
index 6636a97..174144e 100644
--- a/services/jni/com_android_server_BatteryService.cpp
+++ b/services/jni/com_android_server_BatteryService.cpp
@@ -173,6 +173,18 @@ static void setIntField(JNIEnv* env, jobject obj, const
    char* path, jfieldID fie

 static void android_server_BatteryService_update(JNIEnv* env, jobject obj)
 {
+#if TARGET_PLATFORM == armadillo500
+        env->SetBooleanField(obj, gFieldIds.mAcOnline, true);
+        env->SetBooleanField(obj, gFieldIds.mUsbOnline, false);
+        env->SetBooleanField(obj, gFieldIds.mBatteryPresent, true);
+        env->SetIntField(obj, gFieldIds.mBatteryLevel, 4300000);
+        env->SetIntField(obj, gFieldIds.mBatteryVoltage, 4300000);
+        env->SetIntField(obj, gFieldIds.mBatteryTemperature, 30);
+
+        env->SetIntField(obj, gFieldIds.mBatteryStatus, gConstants.
    statusNotCharging);
+        env->SetIntField(obj, gFieldIds.mBatteryHealth, gConstants.healthGood);
+        env->SetObjectField(obj, gFieldIds.mBatteryTechnology, env->NewStringUTF
    ("Li-ion"));
+#else
     setBooleanField(env, obj, AC_ONLINE_PATH, gFieldIds.mAcOnline);
     setBooleanField(env, obj, USB_ONLINE_PATH, gFieldIds.mUsbOnline);
     setBooleanField(env, obj, BATTERY_PRESENT_PATH, gFieldIds.mBatteryPresent);
@@ -192,6 +204,7 @@ static void android_server_BatteryService_update(JNIEnv* env
    , jobject obj)

     if (readFromFile(BATTERY_TECHNOLOGY_PATH, buf, SIZE) > 0)
```

```
        env->SetObjectField(obj, gFieldIds.mBatteryTechnology, env->
            NewStringUTF(buf));
+#endif
 }

 static JNINativeMethod sMethods[] = {
```

7.4.3 Mouse USB come sistema di tracking

Android, nella versione a disposizione, non implementa il supporto a dispositivi mouse come sistemi di tracking.

In rete è presente il progetto *Patch hosting for android x86 support* [3] che con successo ha portato la piattaforma Android in un pc x86 (Asus Eee-PC). Tra le altre modifiche alla piattaforma, è stata pubblicata una patch al sottosistema di Input Android che inserisce la categoria mouse nella serie di dispositivi di tracking abilitati, se gestita attraverso l'interfaccia *Event Interface* (evdev) del kernel.

La patch introduce la classificazione della nuova tipologia di eventi nei file:
frameworks/base/core/java/android/view/RawInputEvent.java e *frameworks/base/include/ui/EventHub.h b/include/ui/EventHub.h*.

```
diff --git a/core/java/android/view/RawInputEvent.java b/core/java/android/view/
    RawInputEvent.java
index 30da83e..4d9a11a 100644
--- a/core/java/android/view/RawInputEvent.java
+++ b/core/java/android/view/RawInputEvent.java
@@ -13,7 +13,8 @@ public class RawInputEvent {
     public static final int CLASS_ALPHAKEY = 0x00000002;
     public static final int CLASS_TOUCHSCREEN = 0x00000004;
     public static final int CLASS_TRACKBALL = 0x00000008;
-
+    public static final int CLASS_MOUSE= 0x00000010;
+
     // More special classes for QueuedEvent below.
     public static final int CLASS_CONFIGURATION_CHANGED = 0x10000000;

diff --git a/include/ui/EventHub.h b/include/ui/EventHub.h
index 3848d8c..280959c 100644
--- a/include/ui/EventHub.h
+++ b/include/ui/EventHub.h
@@ -52,7 +52,8 @@ public:
         CLASS_KEYBOARD      = 0x00000001,
         CLASS_ALPHAKEY      = 0x00000002,
         CLASS_TOUCHSCREEN   = 0x00000004,
```

[3] http://code.google.com/p/patch-hosting-for-android-x86-support/

```
-           CLASS_TRACKBALL     = 0x00000008
+           CLASS_TRACKBALL     = 0x00000008,
+           CLASS_MOUSE         = 0x00000010
        };
        uint32_t getDeviceClasses(int32_t deviceId) const;
```

Ed inserisce le azioni da intraprendere nel caso il mouse generi eventi nei due file:
frameworks/base/libs/ui/EventHub.cpp e
frameworks/base/services/java/com/android/server/KeyInputQueue.java
similmente alle risposte agli eventi generati da dispositivi trackball.

Infine, il server *WindowManager* viene modificato per visualizzare a schermo il puntatore del mouse: nel file:
frameworks/base/services/java/com/android/server/WindowManagerService.java
viene creato un oggetto Surface tale da disegnare a schermo un triangolo translucido che segue la posizione assoluta raggiunta dal puntatore.

La patch *Android-Add-mouse-support-Event-and-Cursor.patch* è allegata a questo documento e deve essere applicata al repository *android-sources/frameworks/base*.

Capitolo 8

Risultati

I risultati del presente lavoro di Tesi vengono qui suddivisi nelle due fasi principali del progetto svolto:

8.1 Piattaforma di supporto per la board Atmark Armadillo 500 nel kernel Linux

La piattaforma di supporto della board Atmark Armadillo 500 per il kernel Linux ottenuta nel capitolo 6 è da considerarsi il massimo risultato ottenibile rispetto allo stato di supporto degli Application Processors i.MX e dei driver presenti nel mainline kernel.

Di seguito, per ogni device presente nella board, è riassunto il livello di supporto raggiunto.

NOR Flash: *Ottimo*. Transfer Rate $\simeq$ 3,3 MB/s
E' gestita tramite un driver standard in kernel. E' stato possibile mantenere lo schema delle partizioni originario, permettendo il possibile salvataggio di un'immagine *ramdisk* nella porzione userspace.

Porte seriali: *Ottimo*.
Le due porte seriali esposte dalla board sono gestite tramite il relativo driver di piattaforma. Il driver in questione fornisce supporto anche a transceiver IrDA per una possibile espansione futura.

Rete ethernet: *Discreto*. Transfer Rate (NFS)$\simeq$ (Min: 200 kB/s, Med: 600 kB/s, Max: 2 MB/s)
Il controller Ethernet è gestito tramite un driver standard in kernel

fornendo piena funzionalità di rete alla board. Non è implementata l'interfaccia verso il sistema di Power Management Linux.

Controller SD/MMC: *Discreto.* Transfer Rate $\simeq$ 5 MB/s
Il controller SDHC dell'Application Processor i.MX31L è ben supportato dal relativo driver di piattaforma fornendo pieno supporto alle schede di memoria SD/MMC. Ciò che limita le prestazioni di tale driver è l'assenza del supporto al sottosistema SDMA nella piattaforma MXC, lacuna discussa in seguito.

Output Video: *Buono.*
L'Application Processor i.MX31L presenta una catena di gestione video molto complessa. Il driver utilizzato ne attiva la porzione strettamente necessaria per una piena funzionalità di video output su display LCD ed ora anche su display analogici tramite video DAC.
Alcune funzioni esposte dal driver devono essere verificate e corrette, mentre il supporto alla catena IPU potrebbe essere maggiormente esteso.

NAND Flash: *Ottimo.* Transfer Rate $\simeq$ 6,5 MB/s
E' gestita tramite il relativo driver di piattaforma capace di supportarla appieno.

Real Time Clock: *Ottimo.*
Grazie al driver di piattaforma di gestione del controller I^2C ed al driver standard in kernel per il chip utilizzato, il sistema può usufruire di un Real Time Clock pienamente funzionale.

Tasti: *Ottimo.*
Gestiti attraverso il relativo driver standard in kernel, sono stati connessi al sottosistema di Input per una piena utilizzabilità nel sistema.

Controller USB: *Buono.* Transfer Rate $\simeq$ 5 MB/s
Grazie alle patch applicate, i due controller USB Host sono stati attivati a piena funzionalità. Il processo di standardizzazione di tali patch è ancora in corso, rendendo questo codice passibile di modifiche nel recente futuro.

Codec Audio: Assente.
Grazie allo stato di supporto in kernel, è possibile dialogare con il codec audio attraverso il bus I^2C e la relativa interfaccia Alsa di gestione low level è presente e funzionale. E' necessario implementare però il driver di collegamento tra l'architettura high level Alsa e

quella low level che utilizzi le funzionalità specifiche del processore (AUDMUX SDMA ..).

Modulo Watchdog: Assente.
Non è presente in kernel un driver di gestione del modulo watchdog dell'Application Processor i.MX31. Questo impendisce all'ambiente user-space linux di operate il reset del sistema.

Sottosistema SDMA: Assente.
Il modulo DMA interno all'Application Processor i.MX31 è implementato attraverso un core μRISC (vedi figura 4.3). Dato che nell'evoluzione degli Application Processors della famiglia i.MX l'architettura interna di tale core indipendente (e quindi l'interfaccia di comando esposta) ha subito modificazioni significative (specie dalla famiglia i.MX2x al i.MX31 ed ancora nel i.MX35), non è presente ancora nel mainline kernel il codice di supporto al modulo SDMA dell'Application Processor i.MX31.
Come risultato, le periferiche esterne che potrebbero usufruire dell'architettura DMA per il trasferimento dati (memory card SD/MMC, ATA, etc.) impegnano in modo importante la CPU.

8.2 Android nella board Atmark Armadillo 500

Nel capitolo 7 è stata generata una versione di Android adatta alla board Atmark Armadillo 500.

Se da un lato si sono risolti i problemi di funzionalità della piattaforma (il boot avviene correttamente presentando a schermo la Home Activity con la possibilità di navigare nel menu e nelle applicazioni installate) ne colpisce la notevole lentezza di risposta.

Certamente la configurazione di test utilizzata (root filesystem montato via nfs) non aiuta la velocità di caricamento delle librerie ed applicazioni allungando i tempi di boot, ma ciò che sovraccarica il sistema è la richiesta notevole di memoria: è stato sviluppato un semplice script di raccolta informazioni a riguardo, basato sul controllo periodico (ogni 2 secondi) delle informazioni contenute in `/proc/meminfo`. Il risultato è presentato nel grafico in figura 8.1.

Si può ben notare come, a piattaforma caricata completamente, la quantità di memoria libera è paricamente nulla e la domanda di memoria della piattaforma Android (Totale della memoria virtuale allocata) si attesta ad

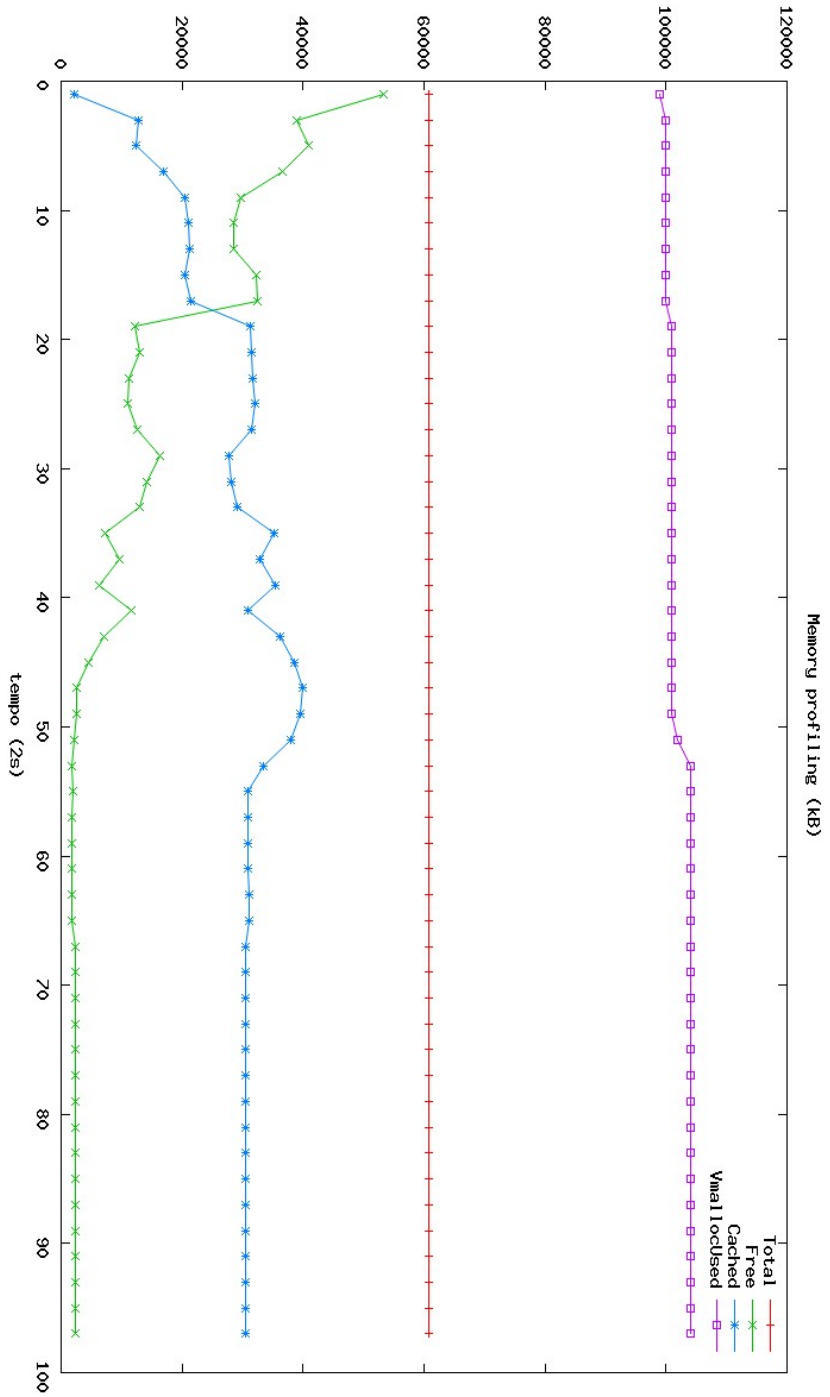


Figura 8.1: Profiling della memoria durante il processo di boot di Android.

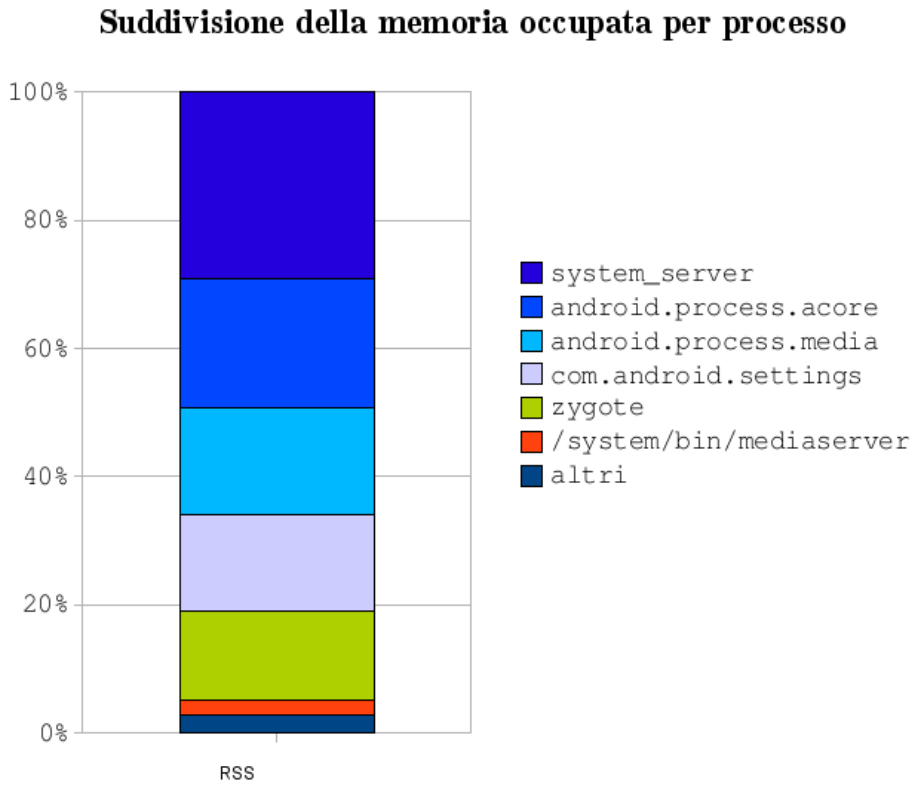


Figura 8.2: Suddivisione Percentuale della memoria occupata, per processo su un totale di circa 114 MB allocati.

un valore circa pari al doppio della Memoria di sistema disponibile. Ogni processo allora è costretto al caricamento del minimo sottoinsieme di librerie possibile per poter eseguire diminuendo notevolmente la capacita di caching del codice in memoria centrale. In questo modo la configurazione di test utilizzata per Android amplifica ancor più il problema dato che il throughput di trasferimento via rete è il peggiore tra quelli valutati.

In figura 8.2 è mostrata la richiesta di memoria per ogni singolo processo in percentuale sulla memoria virtuale utilizzata.

`zygote`: processo padre di ogni applicazione Android. Contiene un'istanza della Virtual Machine Dalvik ereditata in copia dai processi figli. Da solo richiede uno spazio di memoria virtuale pari a circa 18 MB; dimensione minima quindi di ogni suo processo figlio.

`system_server`: è il processo padre di tutto il layer applicativo Android: contiene tutti i server trattati nella sezione 2.2.3 Tra cui *Activity*

Manager e *Window Manager* ed include i server di gestione dei flussi audio e video *SurfaceFlinger* ed *AudioFlinger*.

`android.process.acore`: processo che contiene l'Activity *Launcher*, la *Home* Activity, *Contacts* e le altre activity di base Android.

`com.android.settings`: processo lanciato durante l'esecuzione dell'utility *Settings* nel menu *Home*.

`android.process.media`: processo che contiene il server *Media Provider*, gestore delle risorse di sistema.

Installazione on board. Attraverso la piattaforma GNU Linux creata per i test nel capitolo 6 è stato possibile installare la piattaforma android nel dispositivo flash NAND (`/dev/mtdblock4`).

Il risultato è un miglioramento visibile dei tempi di risposta dovuto all'aumento della velocita di trasferimento tra root-filesystem e la memoria centrale (velocità di caricamento delle librerie a collegamento dinamico), anche se il sistema ancora non si presenta completamente fluido nell'esecuzione.

Capitolo 9

Conclusioni

Questo lavoro di tesi presenta nel dettaglio il know-how necessario per generare ed installare una piattaforma software, dal kernel all'ecosistema user space, in un sistema embedded basato su architettura ARM11.

La board Atmark Armadillo 500 e la piattaforma software Android sono stati dei buoni esempi di come poter applicare queste conoscenze nel caso specifico dimostrando come di volta in volta i problemi riscontrabili e come possano essere risolti giungendo a risultati concreti.

Nell'acquisire il know-how necessario per lo svolgimento di questo lavoro di tesi è stato notevole lo sforzo di ricerca di documentazione specifica riguardo i singoli progetti open source, in special modo:

Kernel API: In determinate porzioni del kernel la documentazione riguardo a particolari meccanismi del kernel o parametri di impostazioni dei driver sono lasciate alla sola analisi del codice stesso, senza che esista documentazione specifica.
Il modello utilizzato dalla comunità eleva a fonte di documentazione la mailing-list di sviluppo, luogo dove si possono ricevere risposte a domande precise che, alcune volte, possono subire difetti rispetto alla realtà implementata.

Android: Il progetto Android dispone di una buona documentazione in rete riguardo l'ambiente applicativo, ma deficita in gran parte della documentazione relativa ad ogni modulo degli strati inferiori: dalle librerie ai moduli kernel.
Come per le kernel API le informazioni vengono scambiate nelle mailing list del progetto dove, in questo caso, non sempre le risposte sono autorevoli.

Riguardo ai risultati ottenuti presentati nel capitolo 8, possono essere fatte le seguenti considerazioni:

Piattaforma di supporto Linux per la board Atmark Armadillo500: Il livello di supporto raggiunto è il massimo possibile dato il tempo e la forza lavoro a disposizione.

Le due lacune importanti (supporto ai sottosistemi DMA e Audio) devono avere la priorità maggiore nell'immediato futuro in seno al completamento del progetto di supporto degli Application Processors i.MX3x nel kernel Linux.

Colmate tali lacune, la board potrà godere di piene prestazioni e funzionalità multimediali.

Android su Atmark Armadillo500: L'architettura sviluppata in Android per l'ambiente applicativo risulta in una richiesta di memoria eccessiva per la board a disposizione: ogni applicazione esegue in seno ad una istanza privata Dalvik ed ingloba tutte le librerie a collegamento dinamico del processo generatore. Il risultatdo è una richiesta di memoria minima di circa 18 MB per applicazione, che sommata ai server del framework Android consuma i 64 MB di memoria disponibile già dopo il caricamento dei core-processes di piattaforma.

Se da un lato la soluzione migliore è l'aumento della memoria di sistema a disposizione (il requisito minimo pubblicato è 128 MB), per migliorare le prestazioni nel breve periodo si propone di implementare un'architettura di swap su una partizione creata ad hoc in un device veloce.

Si propone inoltre di analizzare la compatibilità della nuova revisione del modulo CPU Armadillo 500 FX con la mother board a disposizione: tale modulo CPU, mantenendo lo stesso Application Processor dispone di dimensioni doppie sia della memoria RAM montata sia della memoria Flash NOR.

Appendice A

Memory Map

Armadillo 500

Start Address	End Address	Device	Area
0x0000 0000	0x0000 3FFF	i.MX31 Secure ROM (16KByte)	
0x0000 4000	0x0040 3FFF	Reserved	
0x0040 4000	0x0040 7FFF	i.MX31 ROM Interna (16KByte)	
0x0040 8000	0x1FFF BFFF	Reserved	
0x1FFF C000	0x1FFF FFFF	i.MX31 RAM Interna(16KByte)	
0x2000 0000	0x2FFF FFFF	Reserved	
0x3000 0000	0x7FFF FFFF	i.MX31 Registri dei moduli interni	
0x8000 0000	0x83FF FFFF	DDR SDRAM (64MByte)	CSD0
0x8400 0000	0x8FFF FFFF	Reserved	CSD0
0x9000 0000	0x9FFF FFFF	Reserved	CSD1
0xA000 0000	0xA0FF FFFF	NOR Flash Memory (16MByte)	CS0
0xA100 0000	0xA7FF FFFF	Reserved	CS0
0xA800 0000	0xAFFF FFFF	Regione Extended Bus (128MByte)	CS1
0xB000 0000	0xB1FF FFFF	Reserved	CS2
0xB200 0000	0xB3FF FFFF	Ethernet Controller (LAN9118)	CS3
0xB400 0000	0xB5FF FFFF	Regione Extended Bus (32MByte)	CS4
0xB600 0000	0xB7FF FFFF	Reserved	CS5
0xB800 0000	0xB800 0FFF	NAND Flash Memory	

Start Address	End Address	Device	Area
0xB800 1000	0xB800 4FFF	i.MX31 Registri dei moduli interni	
0xB800 5000	0xBBFF FFFF	Reserved	
0xBC00 0000	0xBFFF FFFF	Regione Compact Flash	
0xC000 0000	0xFFFF FFFF	Reserved	

Elenco degli acronimi

ADC Asynchronous Display Controller.

ADT Android Development Tools.

AHB Advanced High-performance Bus. Tipologia di Bus sviluppata da ARM Ltd parte dello standard AMBA.

AIPS AHB-Lite 2 to IP Bus Rev 3.0 SkyBlue line interface. Interfaccia di collegamento tra il bus AHB-Lite 2 ed il bus Freescale SkyBlue line per periferiche IP compatibili.

AIDL Android Interface Definition Language.

ALU Arithmetic logic unit.

AMBA Advanced Microcontroller Bus Architecture. Standard de facto per le architetture Bus di collegamento tra periferiche[21]. Sviluppato da ARM definisce quattro tipologie di bus: Advanced System Bus (ASB), Advanced Peripheral Bus (APB) evoluto nel Advanced High-performance Bus (AHB) ed infine Advanced eXtensible Interface (AXI). Introdotto nel 1996, la versione più recente è del 2003 AMBAv3.

AP Application Processor. Tipologia di architettura per SoC ottimizzata per dispositivi mobili.

API Application Programming Interface.

ARM Advanced RISC Machines. Compagnia fondata nel novembre del 1990 leader nel settore dei microprocessori e tecnologie affini. Fornisce prevalentemente Intellectual Property (IP).

ARMv6 ARM vesion 6.

APB Advanced Peripheral Bus Tipologia di Bus sviluppata da ARM Ltd parte dello standard AMBA.

ASB Advanced System Bus. Tipologia di Bus sviluppata da ARM Ltd parte dello standard AMBA.

AVIC ARM11-platform Vectored Interrupt Controller. Periferica programmabile per la gestione vettorializzata degli interrupt presente negli Application Processor i.MX31/i.MX31L.

AXI Advanced eXtensible Interface. Tipologia di Bus sviluppata da ARM Ltd parte dello standard AMBA.

BSP Board Support Package.

CCM Clock Control Module.

CLI Command line interface.

CSI Camera Sensor Interface.

DAC Digital to Analog Converter.

DI Display Interface.

DMA Direct Memory Access. Modalità di accesso alla memoria che permette a periferiche abilitate di accedervi direttamente senza il coinvolgimento attivo della CPU.

DPLL Digital Phase-Locked Loop. Dispositivo in grado di sincronizare i fronti di salita (o discesa) di più segnali di clock digitali.

DSP Digital Signal Processor.

EHCI Enhanced Host Controller Interface. Specifiche Intel di Interfaccia per la gestione di controller USB 2.0.

EMI External Memory Interface. Modulo per la gestione della memoria esterna presente negli Application Processor i.MX31/i.MX31L.

EPIT Enhanced Periodic Interrupt Timers.

EPL Eclipse Public License.

FIR Fast Infrared. Nelle connessioni IrDA 1.1 identifica la velocità di trasferimento 4 Mbit/s.

GPIO General Purpose Input/Output. I pin di un processore impostati in modalità GPIO possono essere utilizzati per acquisirne lo stato (alto o basso) o definirne lo stesso; possono inoltre agire come sorgente di interrupt.

GPT General Purpose Timer.

I^2C Inter Integrated Circuit.

I^2S Inter-IC Sound.

IC Image Converter.

IDMAC Image DMA Controller.

IP Intellectual Property. ARM Ltd definisce IP una entità licenziabile (Core di elaborazione, Standerd per Bus di sistema, etc..).

IPC Inter Process Communication.

IPU Image Processing Unit.

IrDA Infrared Data Association.

ISA Instruction Set Architecture. Pate dell'architettura di un elaboratore, specifica il set di istruzioni.

ISV Independent Software Vendors.

LSU Load Store Unit. Unità di esecuzione interna al microprocessore specializzata nelle operazioni di accesso alla memoria.

MCU Microcontroller Unit. Insieme di moduli interni all'Application Processor i.MX31L.

MIR Medium Infrared. Nelle connessioni IrDA 1.1 identifica le velocità di trasferimento 0.576 Mbit/s e 1.152 Mbit/s.

MMU Memory Management Unit.

MPEG Moving Picture Experts Group. Organismo di standardizzazione di codec audio/video.

OEM Original Equipment Manufacturer. Società che acquistano prodotti da altre società e li rivendono come parti di un prodotto proprio.

OSI Open Source Initiative

OTG On The Go. Supplemento allo standard USB 2.0: definisce l'infrastruttura hardware (un nuovo connettore) e software (algoritmi di segnalazione) che permettono al controller USB OTG di agire come Host controller oppure Device controller nella connessione corrente, a seconda del dispositivo ad esso collegato.

PCI Peripheral Component Interconnect. Standard di collegamento per device interni.

PCMCIA Personal Computer Memory Card International Association.

PF Post-Filter.

PHY Physical layer. Abbreviazione di livello fisico: dispositivi o moduli in grado di pilotare fisicamente il canale di comunicazione secondo un determinato standard.

PSRAM Pseudostatic RAM. RAM dinamica che incapsula circuiti di refresh ed accesso, di modo da potervi accedere come SRAM.

RTC Real Time Clock. Modulo hardware capace di mantenere l'orologio di sistema. Attivo anche se il sistema è spento grazie ad una batteria di riserva ed un circuito di clock indipendente.

SDC Synchronous Display Controller.

SDHC Secured Digital Host Controller.

SDK Software Development Kit.

SDMA Smart Direct Memory Access.

SIMD Single Instruction Multiple Data. Aritmetica che permette eseguire con un unica istruzione la stessa operazione su un numero di dati diversi (2-4 ..).

SIR Serial Infrared Speed. Fascia di velocità di trasferimento per connessioni IrDA supportate da una normale porta seriale UART (9600 bit/s, 19.2 kbit/s, 38.4 kbit/s, 57.6 kbit/s, 115.2 kbit/s).

SMC SmartMedia Card. Standard Toshiba per memorie flash.

SoC System on Chip.

SoM System on Module.

SPBA Shared Peripheral Bus Arbiter.

SSI Synchronous Serial Interface.

TCM Tightly-Coupled Memory.

TCP Transmission Control Protocol.

TLB Translation Look-aside Buffer. Tabella di riferimento per la traduzione da indirizzi virtuali a fisici utilizzata dalla MMU per accedere alla cache di sistema.

UART Universal Asynchronous Receiver-Transmitter. Dispositivo capace di gestire I/O seriali come una connessione attraverso la porta RS-232 o IrDA.

ULPI UTMI+ Low Pin Interface. Interfaccia standard per sistemi ad IP USB 2.0 tra link controller USB ed i dispositivi che pilotano effettivamente il bus (transceiver che implementano il livello fisico PHY).

USB Universal Serial Bus. Standard di collegamento per device esterni.

USBH USB Host.

USBOTG USB On-The-Go.

UTMI USB Transceiver Macrocell Interface. Standard Intel di interfaccia verso moduli PHY solitamente posti in-device (link layer e physical layer risiedono nello stesso chip). La versione UTMI+ implementa l'add-on OTG dello standard USB.

RAM Random Access Memory.

VIC Vectored Interrupt Controller.

VFP Vectorial Floating Point. Coprocessore vettoriale per calcoli in virgola mobile. Estensione per i processori ARM implementa lo standard IEEE 754 estendendolo con istruzioni che operano in parallelo su piccoli vettori.

Bibliografia

[1] 2-wire real-time clock s-35390a. Datasheet, Seiko Instruments Inc.

[2] Android build system. Nel repository Android *build/core/build-system.html*.

[3] Armadillo 500 base board parts datasheet. Nel CD allegato alla board, la cartella: */document/hardware/parts*.

[4] Bionic c library overview http://android-platform.googlegroups.com/attach/0f8eba5ecb95c6f4/OVERVIEW.TXT?gda=xLruRUUAAAB1RXoVyyH5sRXFfLYnAq48KOFqr-45JqvtfiaR6gxIj6ksrkgpaf_ULsZG2VEZPWeO3f1cykW9hbJ1ju6H3kglGu1iLHeqhw4ZZRj3RjJ_-A&pli=1&view=1&part=4.

[5] Documentazione android open source project. http://source.android.com/documentation.

[6] The frame buffer device. Nel Kernel root tree: *Documentation/fb/framebuffer.txt* e altri nella cartella.

[7] Gpio interfaces. Nel Kernel root tree: *Documentation/gpio.txt*.

[8] Kernel parameters. Nel Kernel root tree: *Documentation/kernel-parameters.txt*.

[9] Linux driver model. Nel Kernel root tree: *Documentation/driver-model/**.

[10] Linux kernel coding style. Nel Kernel root tree: *Documentation/CodingStyle.txt*.

[11] Linux kernel development process. Nel Kernel root tree: *Documentation/development-process/**.

[12] The linux-usb host side api. DocBook nel Kernel root tree.

[13] Mounting the root filesystem via nfs (nfsroot). Nel Kernel root tree: *Documentation/filesystems/nfsroot.txt*.

[14] Platform devices and drivers. Nel Kernel root tree: *Documentation/driver-model/platform.txt*.

[15] Usb gadget api for linux. DocBook nel Kernel root tree.

[16] Video android developer. `http://developer.android.com/intl/it/videos/index.html`.

[17] Intel strataflash embedded memory (p30) family (pc28f128p30b85). Datasheet, Intel, 2006.

[18] Lan9118 high performance single-chip 10/100 non-pci ethernet controller. Datasheet, SMSC, 2008.

[19] Cmos, 330 mhz triple 8-bit high speed video dac adv7125. Datasheet, Analog Devices, 2009.

[20] Tomi Ahonen. Trillion with a t, the newest giant industry has arrived: the money and meaning of mobile. `http://communities-dominate.blogs.com/brands/2008/12/trillion-with-a.html`, Dicembre 2008.

[21] ARM Ltd. *AMBA Specification*, rev 3.0 edition, 2003. Web: `www.arm.com`.

[22] Atmark. Armadillo 500 base board schematics. Nel CD allegato alla board: */document/hardware/armadillo-500-base-sche_revb1.pdf*.

[23] Atmark. Atmark armadillo 500 board documentation resources. `http://download.atmark-techno.com/armadillo-500/document/`.

[24] Lora Bentley. More on g1 and google's goals with android. `http://www.itbusinessedge.com/cm/blogs/bentley/more-on-g1-and-googles-goals-with-android/?cs=15048`, Settembre 2008.

[25] David Brash. The arm architecture version 6 (armv6). White paper, ARM Ltd, 2002.

[26] Alberto Panizzo Charly Bechara, PeterPearse. Booting arm linux smp on mpcore . http://www.linux-arm.org/LinuxBootLoader/SMPBoot, 2009.

[27] David Cormie. The arm11tm microarchitecture. White paper, ARM Ltd, 2002.

[28] David Cormie. How to participate in the linux community. White paper, The Linux Foundation, 2008.

[29] Elvis Dowson. Re: Is 2.6.29 kernel supported?. http://www.mail-archive.com/android-kernel@googlegroups.com/msg00290.html.

[30] Wikipedia EN. http://en.wikipedia.org/wiki/Dhrystone.

[31] Jonas Fonseca. Git quick reference. http://jonas.nitro.dk/git/quick-reference.html.

[32] USB Implementers Forum. Universal serial bus. http://www.usb.org/home.

[33] Freescale semiconductor. *MCIMX31 and MCIMX31L Applications Processors Reference Manual*, rev 2.3 edition, 2007. Web: www.freescale.com.

[34] Gartner. Gartner says worldwide mobile phone sales declined 6 per cent and smartphones grew 27 per cent in second quarter of 2009. http://www.gartner.com/it/page.jsp?id=1126812, Agosto 2009.

[35] Denton Gentry. The six million dollar libc. http://codingrelic.geekhold.com/2008/11/six-million-dollar-libc.html.

[36] Google. Android application fundamentals. http://developer.android.com/intl/it/guide/topics/fundamentals.html.

[37] Google. The androidmanifest.xml file. http://developer.android.com/intl/it/guide/topics/manifest/manifest-intro.html.

[38] Google. Contribute.
http://source.android.com/download/using-repo.

[39] Google. Using repo and git.
http://source.android.com/submit-patches.

[40] Dan Bornstein (Google). Dalvik vm internals
http://sites.google.com/site/io/dalvik-vm-internals,
2008.

[41] Intel. Ehci specification.
http://www.intel.com/technology/usb/ehcispec.htm.

[42] Russell King. Sending in a patch.
http://www.arm.linux.org.uk/developer/patches/info.php.

[43] Russell King. When i trace kernel in head.s i found problems that may be caused by __turn_mmu_on. any idea ?
http://www.arm.linux.org.uk/mailinglists/faq.php.

[44] Russell King. Booting arm linux.
http://www.arm.linux.org.uk/developer/booting.php, 2004.

[45] Markus Levy. Mpf hosts premiere of arm1136 cores combine armv6, simd, eight-stage pipeline, and more. *Microprocessor Report*, october 2002.

[46] Vincent Sanders. Booting arm linux.
http://www.simtec.co.uk/products/SWLINUX/files/booting_article.html, 2004.

[47] Linus Torvalds <torvalds@transmeta.com>. Re: Availability of kdb.
http://lwn.net/2000/0914/a/lt-debugger.php3, 2000.

[48] International Telecommunication Union. Worldwide mobile cellular subscribers to reach 4 billion mark late 2008.
http://www.itu.int/newsroom/press_releases/2008/29.html.